

Aflevering 1 It 2

Andreas Bach Berg Nielsen s205869

Naveed Imam Shah s205491

Shiv Gopal S205490

Troels Engsted Kiib s205492

Gruppe nr.5

Github: <https://github.com/Troels21/JavaFX.0.2>

February 2021

Abstract

During this IT assignment at DTU, we were assigned to create a java software which implements JavaFX to make a GUI. Our software had some requirements which were; displaying data in javaFx, saving data, displaying saved data, making an alarm system with reconfigurable limits and controlling user accessibility. Furthermore, we had to document our process using Unified Modeling Language and test the GUI prototype on personas. When we were doing this we made 7 diagrams explained as part of UML which represents our thought process. Then we also made a java program based on our UML. During our testing phase we tested the GUI on a Doctor and wrote the feedback down. From this process we learned that some elements in our GUI had to be slightly changed for an improved usability. Therefore, we made some adjustments following the given advice. During this assignment, we learned that we must improve on the DRY and KISS principles while constructing a java code. Overall, we made a working and a functioning program which fulfilled all the given requirements and the requests from our testing phase. Thus we have fulfilled the request from Diamongo.

Indholdsfortegnelse

1	Introduction	4
1.1	Baggrund	4
1.2	Stakeholder analyse	4
1.3	Kort om personaer	5
1.4	Hvorfor er dette projekt relevant?	5
2	Kravspekifikation	6
2.1	Udleveret information fra projektstiller	6
2.2	Løsningsmulighed	6
2.3	Use-case	6
2.4	Non funktionelle krav	7
3	Analyse	8
3.1	Personaer	8
3.1.1	Brugerens Formål	8
3.1.2	Brugerens Interesser	8
3.1.3	Brugerens Forståelse	9
3.2	Hvordan fungerer den nuværende løsning?	9
3.3	Analyseklassediagram	10
4	Design	10
4.1	AktivitetsDiagram	10
4.2	De Forskellige Klasse	12
4.2.1	Main	12
4.2.2	ControllerLogin	12
4.2.3	ControllerProgramChooser	12
4.2.4	ControllerProgramChooserNoAlarm	12
4.2.5	EKGController	12
4.2.6	Beregner	12
4.2.7	ControllerArkiv	12
4.2.8	ControllerAlarm	13
4.2.9	FileHandler	13
4.3	KollaborationsDiagram	14
4.4	DesignKlasseDiagram	14
5	Implementering	16
5.1	Impelementations KlasseDiagram	16
5.2	JavaFX	18
5.3	GUI	19
6	Implementation af JavaKode	21
6.1	Main	21
6.2	ControllerLogin	22
6.2.1	Variabler	22
6.2.2	Metoder	22
6.3	ControllerProgramChooser	25
6.4	ControllerProgramChooserNoAlarm	26
6.5	FileHandler	27
6.5.1	Variabler	27
6.5.2	Metoder	27

6.6	Beregner	29
6.6.1	Variabler	29
6.6.2	Metoder	29
6.7	ControllerAlarm	34
6.7.1	Variabler	34
6.7.2	Metoder	34
6.8	ControllerArkiv	36
6.8.1	Variabler	36
6.8.2	Metoder	36
6.9	ControllerPatientArkiv	39
6.9.1	Variabler	39
6.9.2	Metoder	39
6.10	EKGController	41
6.10.1	Variabler	41
6.10.2	Metoder	42
6.11	PulsSpO2TempController	43
6.11.1	Variabler	43
6.11.2	Metoder	44
7	Afprøvning	45
7.1	Afprøvning på personaer	45
7.1.1	Læge	45
7.1.2	Patient	45
7.2	White box	45
7.2.1	Basis path testing	46
7.2.2	Læge	46
7.2.3	Sundhedspersonale	47
7.2.4	Patient	47
7.3	Path testing	48
7.4	Black box	49
7.4.1	Login	50
7.4.2	ChooseProgramScene	53
7.4.3	Close funktioner	53
8	Diskussion	54
8.1	Sammenligning afprøvningen med personaerne:	54
8.1.1	Lægen	54
8.1.2	Patient	54
8.1.3	Gennemgang af persona-afprøvning	54
8.2	Fejlkilder	55
8.3	Kode optimering	55
8.4	Videreudvikling af projektet:	55
9	Konklusion	56
10	Literaturliste	56
11	Bilag	56

1 Introduction

Teknologi spiller en vigtig rolle i sundhedsindustrien, og den indflydelse har progressivt intensiveret sig. Teknologianvendelse i sundhedsindustrien har længe været med til, at forbedre vilkårene for læger og sundhedspersonalet, men også patienterne, da det i sidste ende er dem, som får en bedre behandling. En af måderne hvorpå teknologi kan forbedre disse vilkår er, at forbedre brugbarheden af medicinsk udstyr og tilgængelighed af information på eksempelvis sundhedsplatformen.

Et konkret eksempel på dette kunne være, et system hvor målinger kan foretages og data kan fremvises. I stedet for at have et overkompliceret system, som udelukkende lægen forstår, så kan man vha. teknologien udvikle noget, som kan det hele. Alt fra at vise alle relevante målinger på en gang og i specifikke tidsintervaller, til at gøre programmet så brugervenligt, at patienten på egen hånd kan benytte det, uden direkte assistance fra et sundhedspersonale. For at kunne introducere selve projektet, og hvad systemet egentlig skal kunne, har vi i introduktionen sat fokus på, at formidle; et formål, en stakeholder analyse, kort om personaer og relevansen til dem.

Der startes med formålet, for at gøre det meget klart, hvad hele projektet egentlig går ud. Efter formålet går vi i gang med stakeholder analysen, som er relevant, til at kunne analysere de aktuelle stakeholders. Hermed kan vi få konkretiseret de relevante aktører og deres indflydelse, samt hvem der er vigtige at tage hensyn til under dannelsen af softwaren. Efter dette, vil vi kort nævne, hvilke personaer vi har med og gøre, som er essentielle for, hvordan systemet skal udvikles. Hvor vi til sidst netop får forklaret relevansen, altså hvorfor netop dette projekt, er en god ide at lave.

1.1 Baggrund

Firmaet Diamongo sælger hardware til måling af patientdata. Diamongo vil gerne komme ind i 2021 med produkter, der kan måle temperatur, puls, iltmætning og EKG. Derfor har de hyret os, til at udvikle softwaren. Vores projektgruppe, er i gang med at tage uddannelsen sundhedsteknologi på DTU, og skriver derfor projekt over opgaven. Vi vil starte med at udarbejde en stakeholder analyse, fordi vi vil redegøre for aktørernes relevans, og for deres grad af indflydelse på projektet.

1.2 Stakeholder analyse

Vi laver en stakeholder analyse, for at danne et overblik over de relevante stakeholders i forhold til produktet. Vi vurderer stakeholdersne ud fra deres magt og indflydelse, samt deres interesse i projektet. Stakeholder analyse ses i figur: 1

Diamongo - Har en stor interesse og indflydelse på produktet. De vil gerne ende med profit. Derfor skal produktet have en mulighed for, at indtjene produktionsomkostninger. De er arbejdsgiveren, så det er dem der stiller kravene.

Projektgruppe - Laver produktet, og har derfor mulighed for at forme produktet. Projektgruppen er derfor både interesseret i, hvordan produktet virker, samt hvor godt produktet er, da de vil fremstå kompetente overfor Diamongo og omverden.

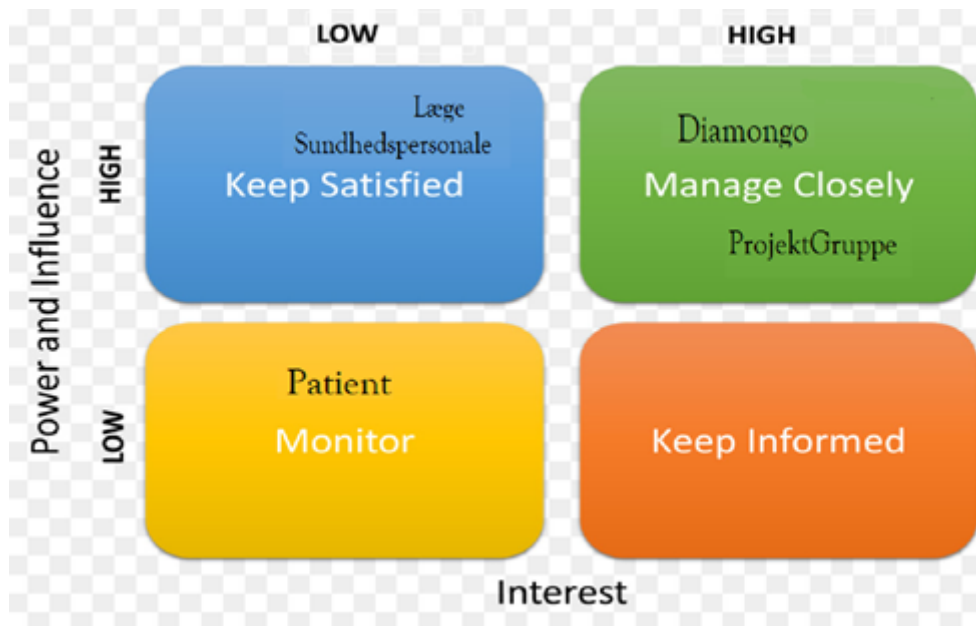


Figure 1: Stakeholderanalyse

Lægen/Sundhedspersonale - Lægen og sundhedspersonalet er ikke interesseret i, hvordan produktet fungerer, men stadig har de en stor betydning for produktet. De er slutbrugerne, og deres tilfredshed med produktet, vil kunne afspejles i Diamongos evne til at få produktet solgt.

Patient - Vores patienter er ligeglade med, hvordan produktet fungerer, og har næsten ingen indflydelse på vores produkt. Patienten skal have så få valgmuligheder som muligt, og skal bare få leveret den brugbare information uden videre.

1.3 Kort om personaer

For at kunne udvikle det mest brugervenlige system, er det relevant at undersøge de personaer, som udgør systemets brugere. Derfor vil vi udføre en analyse af personaerne, så programmet kan designes og justeres til personaerne, så systemet er brugervenligt for dem. De 3 personaer som udgør brugergruppen er hhv: Lægen, sygeplejersken og patienten. Normalt er der mange aspekter som fx brugerens personlighed, præferencer osv., men dette system skal anvendes i sundhedsvæsenet. Derfor er der nogle andre aspekter, som er centrale for personaerne. Dette har vi vurderet til at dreje sig om: Brugerens formål, brugers begrænsninger, brugerens forståelse. Disse vil vi analysere på i selve analysen.

1.4 Hvorfor er dette projekt relevant?

Projektet er relevant, fordi sensor modulerne som Diamongo producere, skal indeholde et stykke relevant kode, således at det sundhedsfaglige personales opgaver bliver lettere, og patienternes oplevelse bliver bedre. Derfor skal softwaren have en høj Usability. Diamongo vil gerne have et godt stykke software, så deres firmas omdømme bliver bedre. Dog har de et forholdsvis lille budget. Derfor har de fået vores projektgruppe til, at lave softwaren, da projektgruppen er forholdsvis ny, og derfor billige at hyre. For projektgruppen er projektet relevant, da det giver

en erfaring med brugen af JavaFX, rapportskrivning og Java. Dernæst får vi til slut et bedre omdømme, således at vores projektgruppe kan få flere og bedre projektmuligheder.

2 Kravspecifikation

I dette afsnit skal vi have kigget på de krav som vi bliver stillet til vores projekt. Dette inkluderer en problemstilling, samt løsnings muligheder, tilhørende use-case diagram og en liste over non funktionelle krav.

2.1 Udleveret information fra projektstiller

Softwaren til deres sensorer skal bruges, til at opsamle temperatur, iltmætning, puls og EKG. Softwaren skal kunne lagre og genkalde dataene, samt se en visning af dataene. Man skal kunne se flere data typer i samme visning og have mulighed fra at vælge hvilken data der skal vises. Sundhedspersonalet skal kunne tilgå deres alt dataene, imens patienten kun skal kunne tilgå sin egen. I softwaren skal der kunne indstilles alarmgrænser for de forskellige måleparametre af læger. Det er "nice to have", hvis lægerne kan kommentere på hændelser ud fra tidspunkt og målingstype.

2.2 Løsningsmulighed

Patient, Sundhedspersonalet og Lægen, vil kunne tilgå et modul, hvori GUI der indeholder en Login side. De vil hver især have fået et login, hvormed de vil kunne tilgå hver deres respektive valg muligheder. Patienter skal kunne se deres logge ind med deres CPR-NR. og kun kunne se deres egen data. Sundhedspersonalet skal kunne lave undersøgelsen og logge ind via. Deres givende brugernavn og se undersøgelsens data på alle patienter. Læger skal kunne alt det samme som sundhedspersonalet, men yderligere have mulighed for at sætte alarmgrænser for målingerne.

2.3 Use-case

I forhold til vores løsningsmulighed, har vi konstrueret et Use-case diagram. Use-case diagram ses i figur: 2.

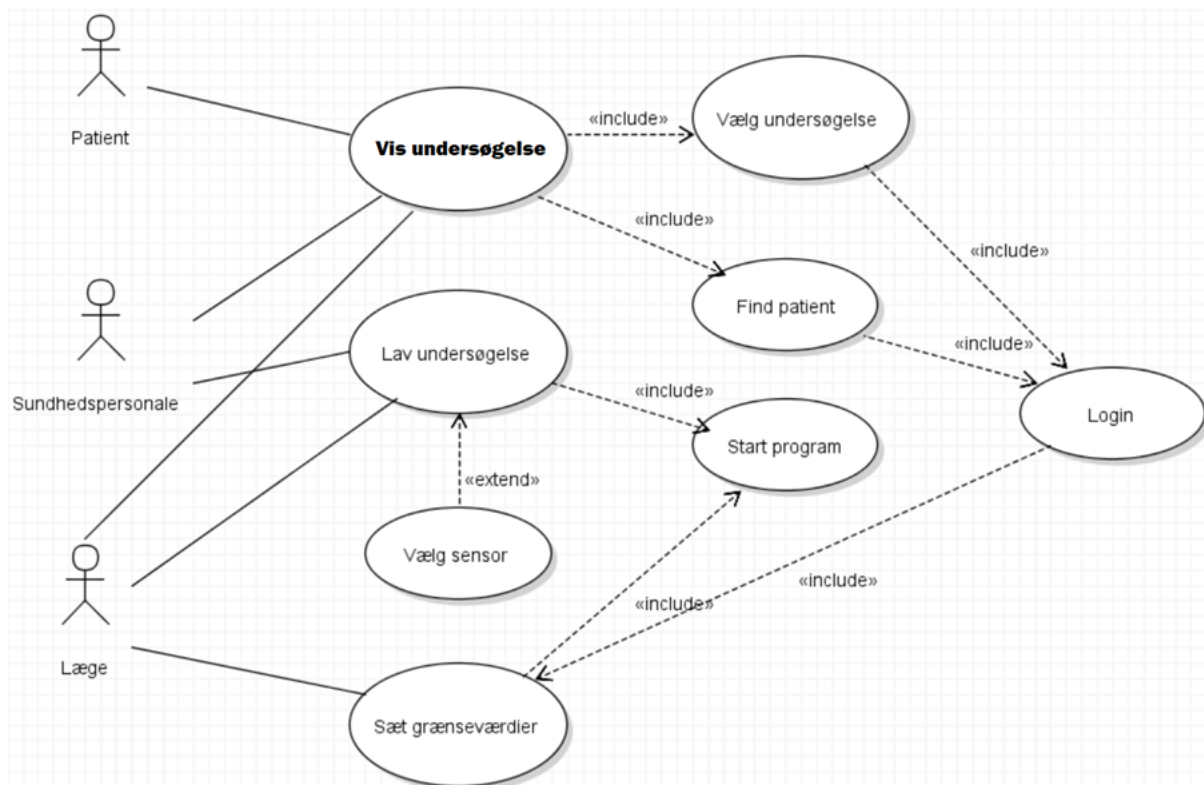


Figure 2: Use-case Diagram

2.4 Non funktionelle krav

Det skal være lavet i javaFX og senere benytte en sql database til at lagre opsamlede målinger. Dokumentere arbejdsproces, arbejdsfordeling, anvendte værktøjer: IntelleJ, VioletUML, Gluon Scenebuilder, Microsoft Office, Overleaf LaTeX, Github.

3 Analyse

I dette afsnit har vi udarbejdet personaer, til at repræsentere de brugergrupper, der blev beskrevet i figur 2 under kravsspecifikationen. De er lavet over en læge, en sygeplejerske og en patient. Herunder beskrives deres formål, tekniske kompetencer og baggrund, for at give et bud på, hvad der skal designes til. En teknisk baggrund behøver ikke at være en ingeniøruddannelse, det kan også være den teknologi, som brugeren anvender til daglig. Alt fra en smartphone og formålene med dem, såvel som en tablet til at vise bageopskrifter på, til en insulinpumpe med tilhørende mobilapplikation. Personaerne ses i afsnit 3.1

For at danne overblik over problemområdet, og danne et billede af de fysiske objekter i situationen, anvendes der analyseklassediagrammer i figur 3, til at vise relationer mellem objekterne. Herunder patienter, sundhedspersonale og abstrakte koncepter, såsom en database.

3.1 Personaer

Tidligere blev de relevante personaer opskrevet, og nu analyseres de enkelte personaer ift. de relevante aspekter. Dog er det vigtigt lige at nævne, at det nok er en atypisk måde at udføre personaer på, idet at mange aspekter som personlighed og alder osv. bliver udeladt. Dette har vi gjort bevidst, idet at aspekterne, som vi mener er relevante, er fx brugernes formål med programmet og den teoretiske forståelse brugeren besidder på forhånd, idet at programmet jo netop skal tilrettelægges til et hospitalsmiljø, hvori at sundhedspersonalet og patienterne bogstaveligtalt kan være alle og enhver. Derfor kan personaerne ikke udarbejdes på den typiske måde, som til livstilsapplikationer eller lignende, men at der er mere fokus på deres formelle ”roller”.

3.1.1 Brugerens Formål

Her sættes der fokus på brugergruppernes formål med brugen af systemet:

Lægen - Formålet med anvendelse af systemet for lægen, er at kunne foretage målinger af diverse patienter, og derudover også at kunne få et præcist overblik over resultaterne, fra de foretagende målinger. Udover dette, så vil lægen også være i stand til at sætte alarmgrænser.

Sygeplejersken - Formålet for sygeplejersken er, at kunne foretage og se et præcist overblik over resultaterne, fra de foretagende målinger.

Patienten - Patientens formål er at kunne se sine egne målinger.

3.1.2 Brugerens Interesser

Her sættes der fokus, på hvad brugergrupperne vil med systemet, og hvad der ikke er interessant for dem. Derfor kan deres adgang til de funktioner så begrænses, så systemet bliver mere simpelt og brugervenligt.

Lægen - Lægen bør ikke have nogle systematiske begrænsninger, idet lægen både er interesseret i at kunne sætte alarmgrænser, foretage målinger og se målinger.

Sygeplejersken - Sygeplejersken kan det samme som lægen, men er ikke interesseret i at sætte alarmgrænser, idet det kun er lægen, som sætter dem.

Patienten - Patientens interesse ligger i, at tjekke sin egen data. Så vedkommende vil ikke se andres målinger eller bruge funktionen til at kunne foretage målinger. Så dette bør ikke vises.

3.1.3 Brugerens Forståelse

Her sættes der fokus på brugervenlighed. Idet der er forskel på den sundhedsvidenskabelige viden, som brugergruppen har.

Lægen - Lægen har på baggrund af sin uddannelse, den højeste viden inden for dette. Derfor også en god forståelse, så der ikke skal tilføjes nogle reelle krav til, at noget skal "simplificeres".

Sygeplejersken - Sygeplejersken har også en viden fra sin uddannelse, som gør at der ikke er relevante forsimpelingsbehov i dette sammenhæng. Dog er sygeplejersken ikke egnet til at sætte alarmgrænser.

Patienten - Patienten har som udgangspunkt ingen faglig viden indenfor emnet, og kræver derfor en vis hjælp for at forstå sin data.

For at kunne øge brugervenligheden, vil vi så udvikle systemet således, at datafremvisningen bla. er anderledes for de enkelte personaer. Dette er med til at sørge for, at vi sikre os at programmet er forståeligt, og kun viser det relevante, helt tydeligt. Så ift. forståelsen, laves der en separat scene til patienten, hvori de sundhedsmæssige termer simplificeres markant ift. interesser/begrænsninger/forståelse. Derudover kan patienten kun se sig egen data, og det er både pga. patientens formål, og andre patienters privat liv. På denne måde kan personaerne anvendes til et groft overblik, som kan bruges til implementationen ift. hvordan systemet bedst kan tilpasse sig, til personaerne.

3.2 Hvordan fungere den nuværende løsning?

Denne løsning, som der er blevet fremstillet, virker tilsyneladende meget godt i forhold til de krav, der er blevet stillet. Vi har et funktionsdygtigt program, der kan tage simulationernes værdier, og gemme dem til deres tilsvarende CPR-numrer. Vores brugere er i stand til at logge ind med deres egne logins. Lægerne og sygeplejerskerne har hver deres, samt patienternes login er deres CPR-nummer. Den er brugervenlig, da den har få knapper, samt er dens funktioner simple og lige til. Derfor er vores løsning, en fin løsning til vores fremstillede problem. Nogle forbedringer der kunne laves til vores løsning, er implementationen af SQL databaser. Der er blevet gjort nogle overvejelser til ændring af programmet, såvel SQL databasen blev implementeret. Dette vil være ændring i datalagring, hvor vi så vil gemme data i en database. Dernæst vil vi kunne gemme en masse brugere i databasen, i stedet for vores nuværende løsning, hvor de er skrevet, som en del af koden. Dette vil gøre, at vi så ville være i stand til at oprette brugere, da de så vil blive gemt i databasen, og vil kunne tilgås næste gang vi kører koden. Udover dette, vil der også kunne laves mere struktur i vores metoder, f.eks. at se på opdeling af metoderne, og at fjerne kode fra vores kontroller klasser, og skrive dem separat. Dette vil gøre, at vi i vores kontroller klasse, kun har metodekald.

3.3 Analyseklassediagram

Følgende analyseklassediagram anvendes til at visualisere, hvordan vi gerne vil have, vores program ser ud. Her tages der højde for de overvejelser, der blev foretaget. Der vises bla. funktionaliten for forskellige logins, i og med at programmet er tilrettelagt flere typiske brugere. Altså de tidligere beskrevne personaer, som så kan tilgå forskellige funktioner inde på programmet. Derudover vises der, hvordan vi forventer at kunne lave et program, som hhv. kan måle, gemme og vise data fra diverse patienter. Dette analyseklassediagram ses i figur: 3

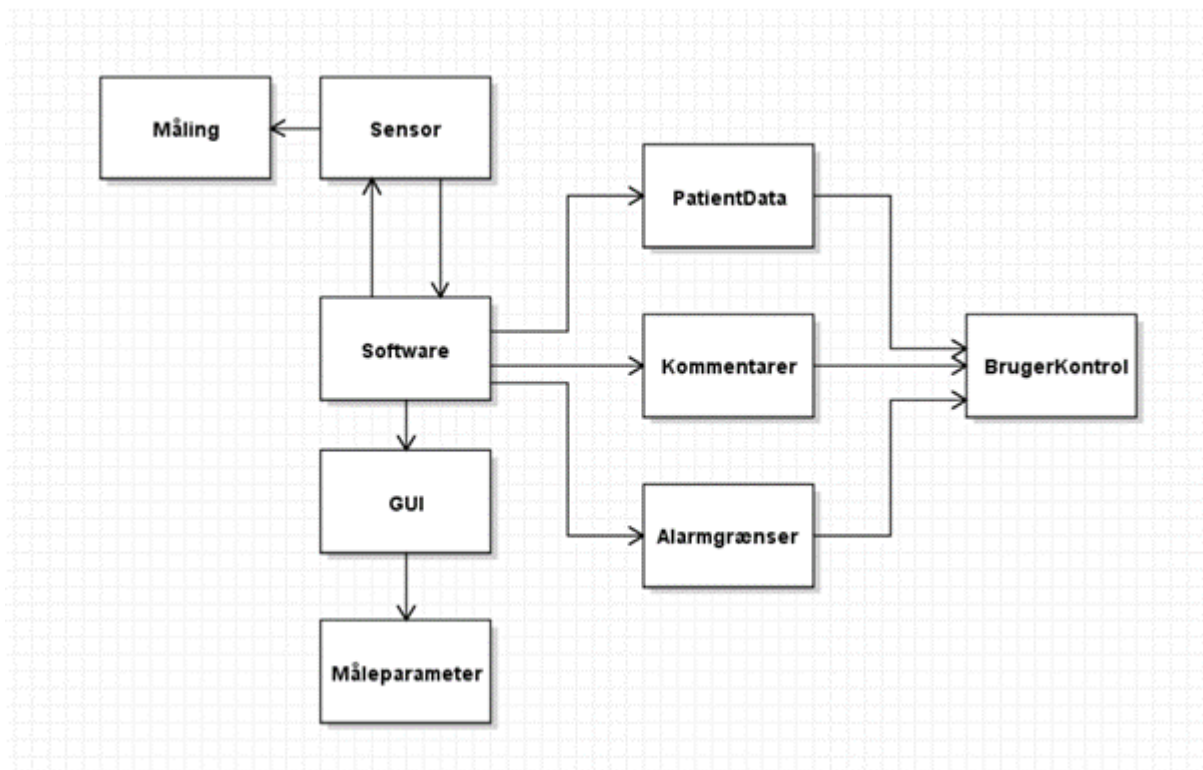


Figure 3: AnalyseKlasseDiagram

4 Design

I dette afsnit skal vi påbegynde dannelsen af vores løsning, på baggrund af hvad der vi kom frem til i analysen. Dette indebærer, at lave både struktur- og opførselsdiagrammer.

4.1 AktivitetsDiagram

Aktivitetsdiagrammet beskriver hvordan programmet skal eksekveres. Først skal man logge ind, hvorefter man bliver ført til sin respektive scene. Patienten vil blive ført til en scene, hvori patientens data bliver fremvist. En læge vil blive ført til sin respektive scene, hvor lægen så kan vælge at klikke på 4 knapper. Sundhedspersonale har kun 3 knapper at vælge imellem, da de ikke skal kunne sætte alarmgrænser. Dernæst kan man få vist eller opdateret data, og til slut står man med valgmuligheden om, at man vil gå tilbage til sin respektive scene eller slukke programmet. Aktivitetsdiagrammet ses i figur:

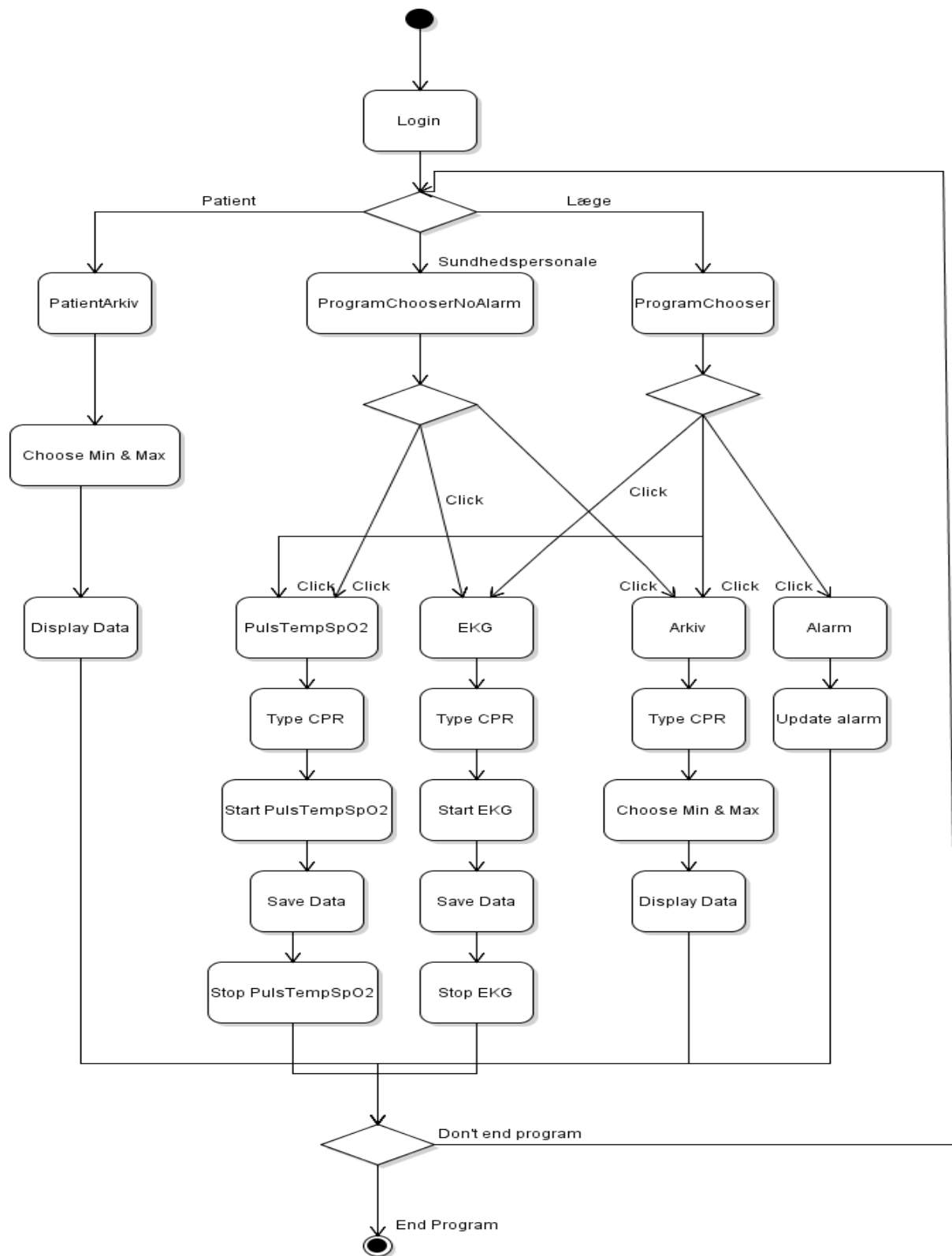


Figure 4: AktivitetsDiagram

4.2 De Forskellige Klasse

4.2.1 Main

Denne Klasse har vi tænkt os at bruge til, at fremvise en scene, hvor vi dernæst gør brug af de andre klasse derhen. Det vil sige vores, at vores Main klasse egentlige bare "initializer" programmet. Dernæst bliver resten kørt af andre klasser.

4.2.2 ControllerLogin

Denne klasse er tilgængelig for alle brugere. Den bruges som kontrol, så den respektive bruger kan logge ind, og at brugeren kun får adgang til det som "login'et" er egnet til. Dette gøres ved kontrolstrukturer, som validere brugerens login, og dernæst viderefører dem til deres respektive side.

4.2.3 ControllerProgramChooser

Dette vil være klassen, som viser den side lægen har adgang til. Denne klasse vil have adgang til alle muligheder, såsom; at tage målingerne, se målingerne og gemme alarmgrænserne.

4.2.4 ControllerProgramChooserNoAlarm

Dette vil være klassen, som sygeplejersken vil have adgang til. Denne klasse kan nærmest det samme som lægens klasse. Sygeplejerskens klasse, har nemlig ikke have adgang til Alarm.

4.2.5 EKGController

Denne klasse plotter et EKG signal, som laves i Beregner klassen, der fremvises på en scene som en realtime simulation.

4.2.6 Beregner

Dette er klasse klassen, hvor simuleringerne af målingerne dannes. Dette indebærer følgende parametre: Temperatur, puls, SpO2 og EKG.

4.2.7 ControllerArkiv

Denne klasse er vores arkiv klasse. Den skal sørge for at læse de gemte filer af filehandler klassen, og fremvise dem på dens scene. Den skal kunne skelne mellem CPR-numrene, og vise målingerne for et bestemt tidsinterval.

4.2.8 ControllerAlarm

Denne klasse har kun lægen adgang til, og den bruges til at opdatere alarmgrænserne for målingerne.

4.2.9 FileHandler

Denne klasse bruges til at gemme alle vores målinger til en fil, som er navngivet efter patientens CPR-nummer.

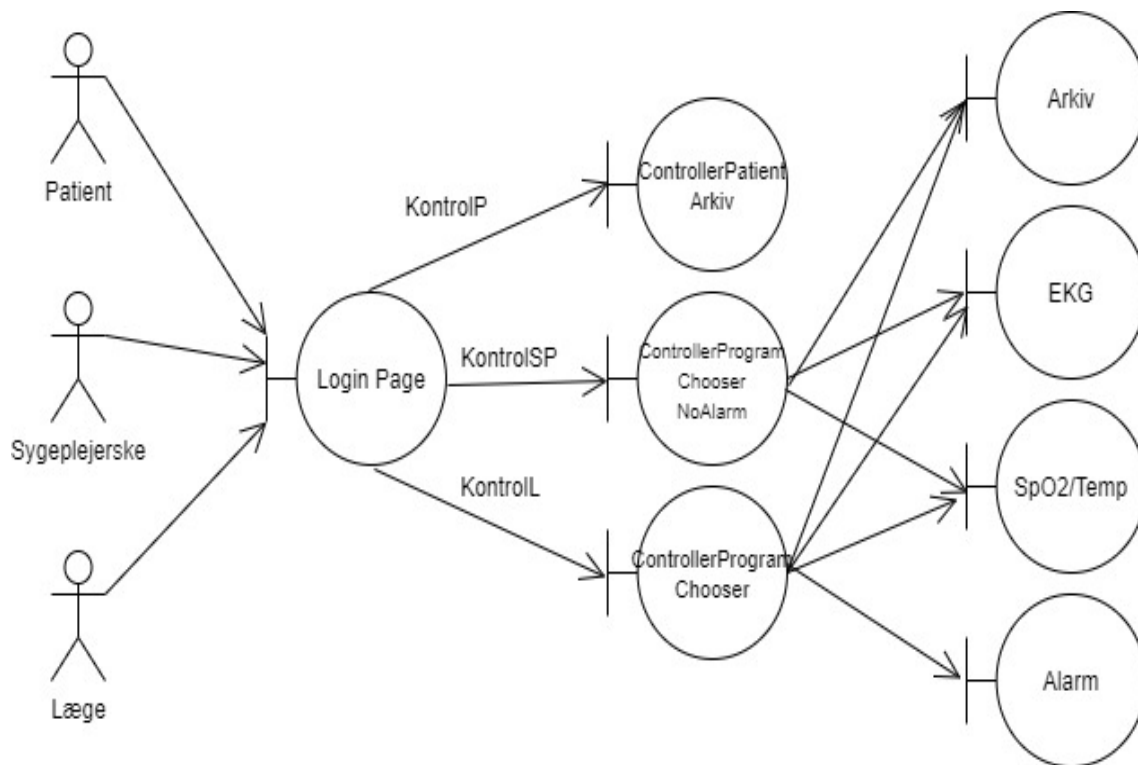


Figure 5: KollaborationsDiagram

4.3 KollaborationsDiagram

Kollaborationsdiagrammet viser, hvordan de forskellige brugere interagerer med programmet. Patienten vil kun have adgang til sit eget arkiv, sygeplejersken vil have adgang til måling af data og arkiv, og lægen vil have adgang til at måle data, sætte alarmgrænser, og se arkivet. På denne måde, bliver der skelnet mellem hvem der bruger programmet. Dette gør programmet mere brugervenligt, da det er designet specifikt til de forskellige brugere. Kollaborationsdiagrammet ses i figur: 5

4.4 DesignKlasseDiagram

Designklasse-diagrammet viser, hvordan vores klasser er ment at se ud. Dette indebærer, de attributter vi bruger, samt de metoder vi gør brug af. Her kan der ses, hvordan hele programmet ligger bag en loginklasse. Dette gør at vi på baggrund af login, kan lave separate sider til brugerne. Dernæst kan det ses, hvordan de forskellige klasser har adgang til yderligere klasser. Hvor patient-Arkiv ikke har adgang til andre end den selv, samt ProgramChooserNoAlarm ikke har adgang til alarm. På denne måde sikrer vi, at der kun opnås adgang til det, som skal bruges af den enkelte bruger. Designklasse-diagrammet ses i figur: 6

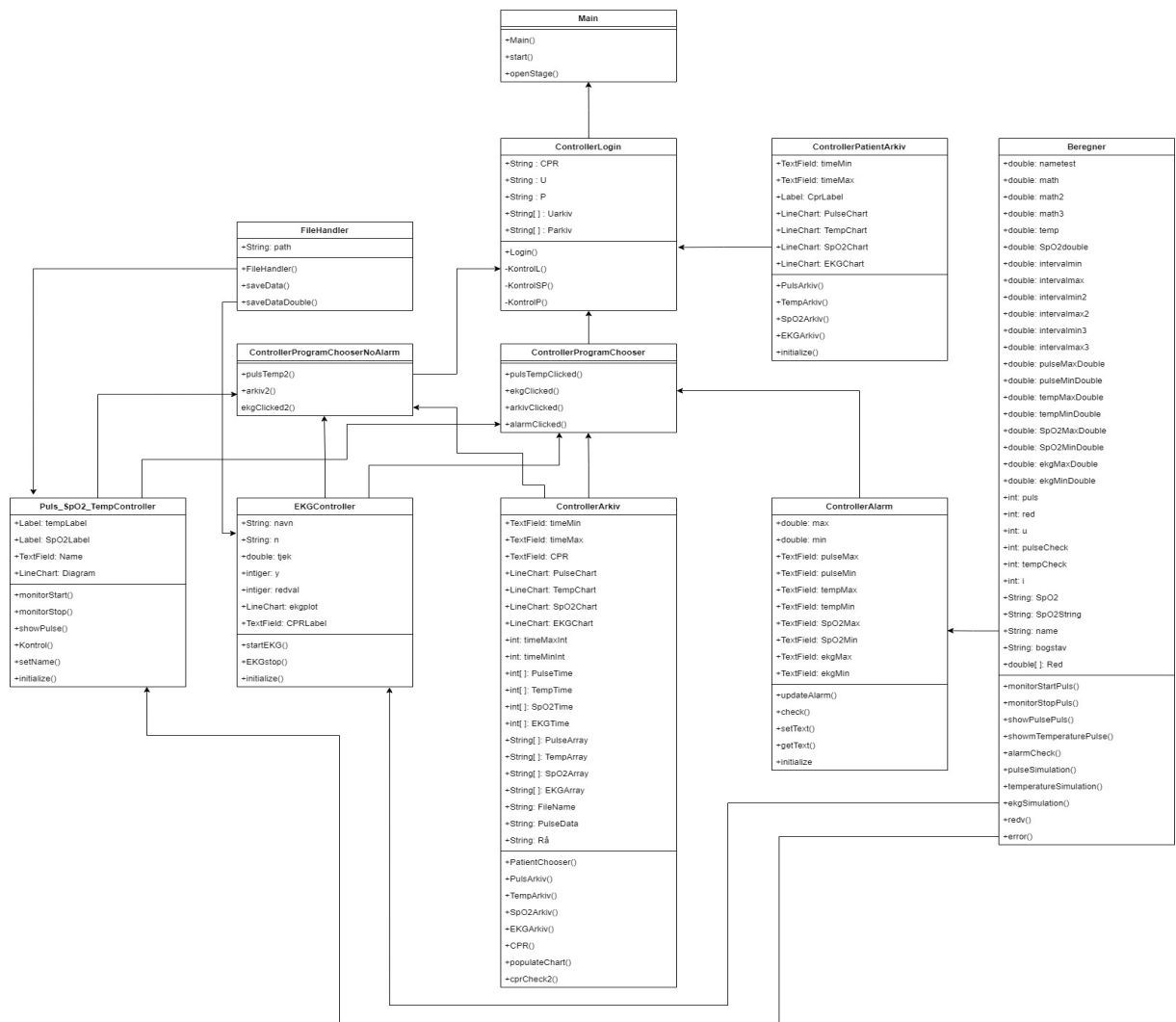


Figure 6: DesignKlasseDiagram

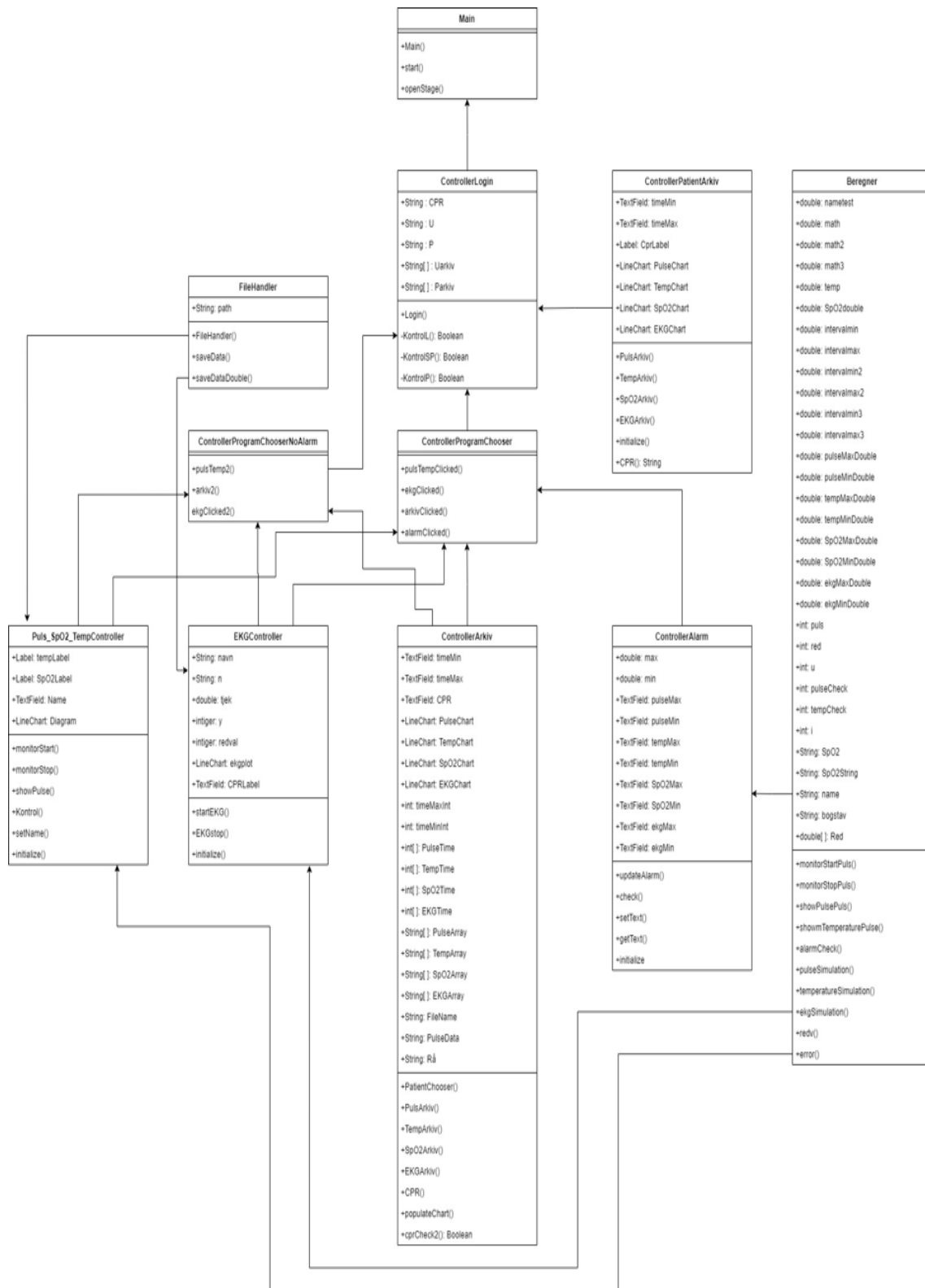
5 Implementering

I dette afsnit vil vi komme ind på vores endelige løsning. Her vil vi kigge på vores løsning, og hvordan den er blevet implementeret i virkeligheden, med et implementationsklassediagram.

5.1 Implementations KlasseDiagram

Dette er vores endelige klassediagram af koden. Denne kode viser, hvordan vi havde overvejet, at vores programs struktur skulle være. Dette afspejler vores analyseklassediagram, hvilket viser et program, som opfylder vores krav. Implementationsklassediagrammet ses i figur: 7

Løsningen er udviklet i Javakode, skrevet med JavaFX i intellij IDE.



5.2 JavaFX

Vi har designet alle UI med GluonHQ scenebuilder. Eksempel på sceneopbygning ses i figur: 8

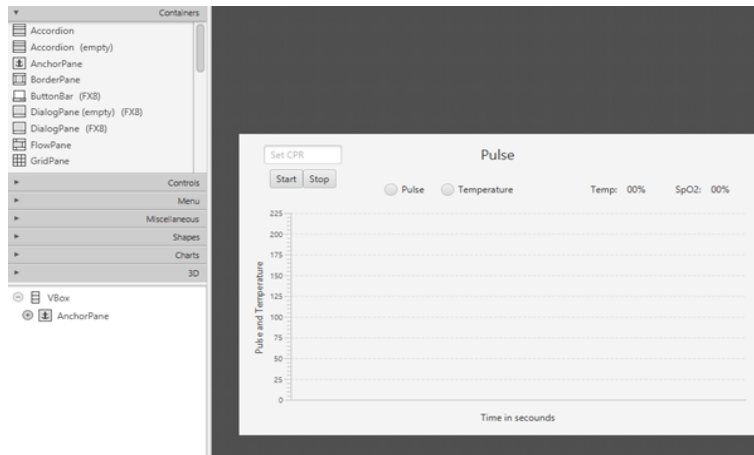


Figure 8: Gluon HQ scenebuilder

Vi har designet scener i Gluon Scenebuilder. Gluon giver os så denne kode, i form af en fxml fil, som vi også manuelt kunne have skrevet. Værktøjet gør det meget nemmere og dynamisk, at designe og ændre i scenerne. På figur: 9 ses den tilhørende fxml fil, fra den før viste scene.

```
<?xml version="1.0" encoding="UTF-8" ?>
<VBox prefHeight="400.0" prefWidth="640.0" xmlns="http://javafx.com/javafx/15.0.1" xmlns:fx="http://javafx.com/fxml/1" fx:controller="se.kth.javafx.controllers.Pulse">
  <children>
    <AnchorPane maxHeight="1.0" maxWidth="1.0" prefHeight="1.0" prefWidth="1.0" VBox.vgrow="ALWAYS">
      <children>
        <Label alignment="CENTER" layoutX="299.0" layoutY="14.0" style="font-size: 18pt;" text="Pulse" textAlignment="CENTER" wrapText="false" />
        <Font size="18.0" />
        </Label>
        <LineChart id="Diagram" fx:id="Diagram" animated="false" layoutX="8.0" layoutY="89.0" prefHeight="304.0" prefWidth="632.0" />
        <CategoryAxis label="Time in seconds" prefWidth="537.0" side="BOTTOM" />
        </CategoryAxis>
        <NumberAxis label="Pulse and Temperature" side="LEFT" upperBound="220.0" />
        </NumberAxis>
        </LineChart>
        <Button layoutX="18.0" layoutY="47.0" mnemonicParsing="false" onAction="#monitorStart" text="Start" />
        <Button layoutX="79.0" layoutY="47.0" mnemonicParsing="false" onAction="#monitorStop" text="Stop" />
        <TextField fx:id="Name" layoutX="31.0" layoutY="15.0" prefHeight="25.0" prefWidth="90.0" promptText="Set CPR" />
        <Label layoutX="540.0" layoutY="65.0" text="SpO2:" />
        <Label fx:id="SpO2Label" layoutX="584.0" layoutY="65.0" text="00%" />
        <RadioButton layoutX="250.0" layoutY="65.0" mnemonicParsing="false" onAction="#showTemperature" text="Temperature" />
        <RadioButton layoutX="180.0" layoutY="65.0" mnemonicParsing="false" onAction="#showPulse" text="Pulse" />
        <Label layoutX="435.0" layoutY="65.0" text="Temp:" />
        <Label fx:id="tempLabel" layoutX="480.0" layoutY="65.0" text="00%" />
      </children>
    </AnchorPane>
  </children>
</VBox>
```

Figure 9: FXML file

Her kan alle de konkrete indstillinger ses, samt hvordan koden er struktureret. Her kan også ses, hvordan knapper og tekst felter hænger sammen med Java koden. Tekst felterne identificeres med fx:id = "name" og tilgås i Java koden. Knapperne har et punkt, som hedder onAction = "metode" som er måden hvorpå, vi giver knapperne funktionalitet.

5.3 GUI

Vores Grafikse brugerflade kan ses på figurene i kapitel 5.3.

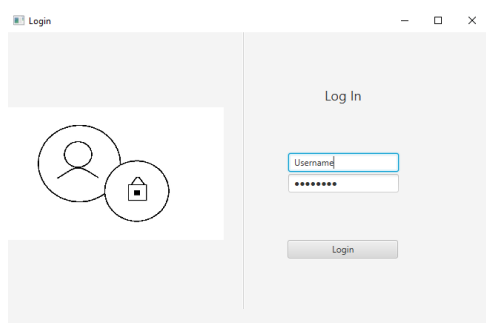


Figure 10: Login scene

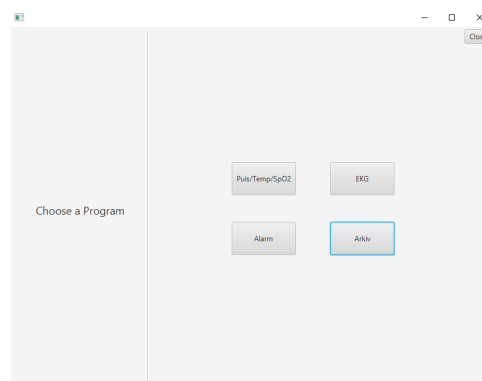


Figure 11: ControllerProgram-Chooser scene

Vi har valgt at designe vores login side, så den stemmer overens med normerne for en loginside. Dette gøres så brugeren nemt forstår hvad de skal. Vi har valgt, at loginsiden skal have både brugernavn samt kodeord, så vi kan differentiere mellem typer af login, men også lave restriktioner på tilgængeligheden. Vi har lavet tre forskellige sider, som kan tilgås via forskellige login navne og tilhørende kodeord. Den første side er den som lægen har adgang til, og den har fire knapper, som har adgang til de fire scener vi har bygget. Det specielle ved denne er, at den har adgang til Alarm. Disse scener kan ses på figur: 10 og 11.

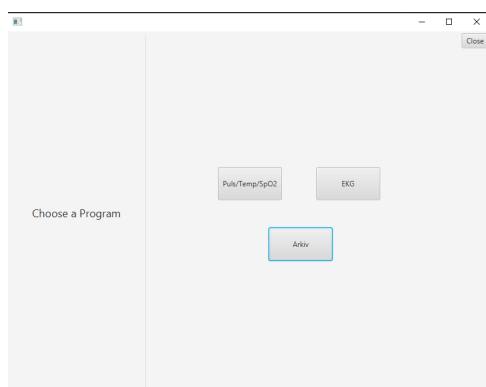


Figure 12: ControllerProgram-ChooserNoAlarm scene

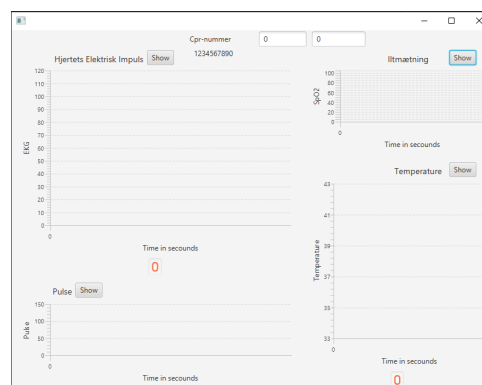


Figure 13: PatientArkiv scene

Her ses siden til sundhedspersonale, hvor Alarm siden er fjernet, så det ikke kan ændre alarmgrænser. Derefter ses siden tiltænkt til patient, som man tilgår ved at logge ind med et cpr nr. som brugernavn. Her får man så vist en scene til Arkiv, dog kan der kun vises data for det cpr nr., brugeren er logget ind med. Disse scener kan ses på figur: 12 og 13.

Her ses den side som åbnes, når sundhedspersonale eller en læge trykker på Puls/temp/spo2 knappen. Vi har designet den så man kan vælge, hvad man vil plotte på grafen. Dernæst kan man så indtaste et cpr nr., som denne data så vil blive gemt under. Startknapper får programmet til at kontrollere, om det indtastede cpr er valid, hvorefter det henter den data, og plotter det.

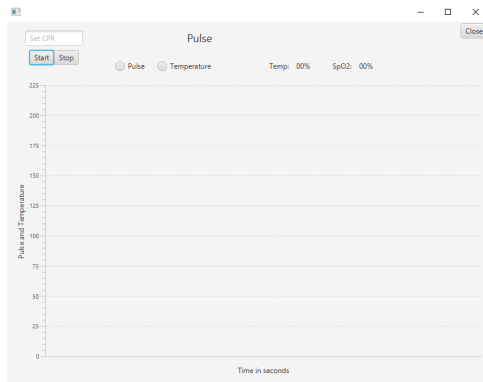


Figure 14: PulseSpO2Temp scene

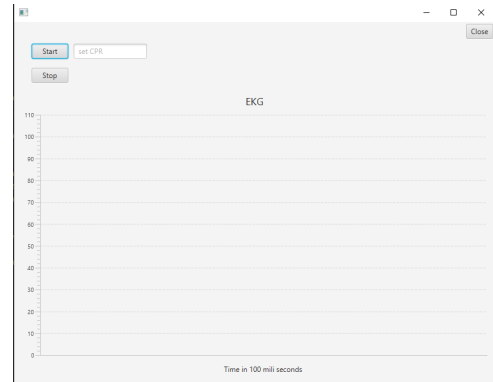


Figure 15: EKG scene

EKG siden fungerer på samme måde, dog er der kun en variabel. Disse scener kan ses på figur: 14 og 15.

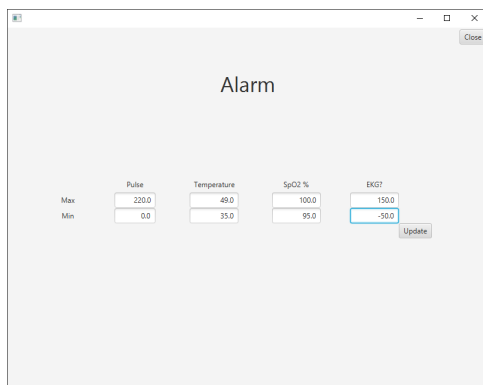


Figure 16: Alarm scene



Figure 17: Arkiv scene

Alarm siden indeholder grænseværdier for alle vores data grupper, som kan ændres ved at trykke på update knappen. Sidst kan man se arkiv siden, som viser data til et tilsvarende cpr nr., og her er der mulighed for at vælge en periode man vil se data i. Disse scener kan ses på figur: 16 og 17.

6 Implementation af JavaKode

Nu har vi gennemgået vores implementation af GUI, som vi konstruerede med Gluon. Nu vil vi gennemgå implementationen af den java kode, som er den bagvedlæggende funktionalitet, der muliggør brugen af vores GUI. Java koden er udarbejdet i IntelliJ, og består af en række kontroller klasser tilhørende til hver deres fxml fil, samt yderligere klasser der inderholder metoder, der giver funktionalitet.

6.1 Main

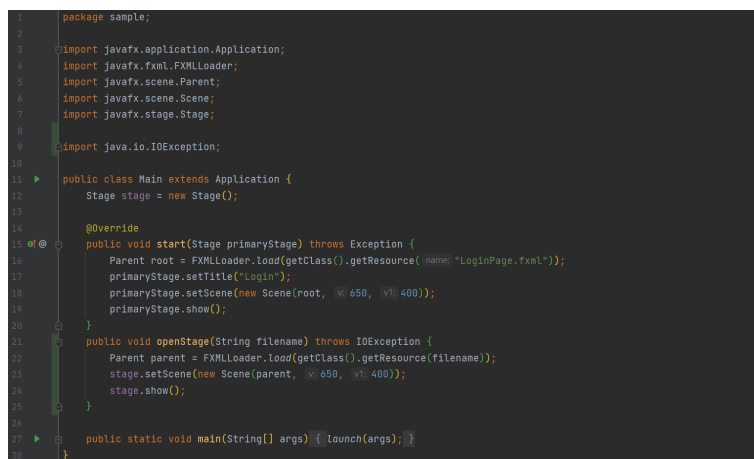
The image shows a screenshot of a code editor with Java code. The code is for a class named 'Main' which extends 'Application'. It includes imports for 'javafx.application.Application', 'javafx.fxml.FXMLLoader', 'javafx.scene.Parent', 'javafx.scene.Scene', 'javafx.stage.Stage', and 'java.io.IOException'. The 'Main' class has a 'start' method that overrides the one in 'Application'. This method loads an FXML file 'LoginPage.fxml', sets the title to 'Login', sets the scene with a width of 650 and height of 400, and shows the stage. There is also an 'openStage' method that does similar actions for a given filename. Finally, there is a 'main' method that calls 'launch'.

Figure 18: Main

Main metoden har til formål at starte programmet, og åbne den første stage. Det åbner vi ved at lave en konstruktør til main metoden, som kalder launch metoden. Vi ekstender Application Klassen, og overrider dens start metode. Netop denne start metode bliver kaldt af launch metoden, og starter derfor programmet. Inde i start metoden skal vi derfor lave en Parent root, som bruger FXMLLoader til at hente "loginpage.fxml" filen. Derefter bruger vi så denne, til at sætte scenen dvs. indstille hvordan vores primarystage, skal se ud. Til slut findes der en close metode, som gemmer stagen, så den "lukker" uden at lukker. Koden kan ses i figur: 18.

6.2 ControllerLogin

6.2.1 Variabler

```
1 package sample;
2
3 import javafx.fxml.*;
4 import javafx.scene.control.PasswordField;
5 import javafx.scene.control.TextField;
6
7 import java.io.File;
8 import java.io.IOException;
9
10 public class ControllerLogin {
11     Main m = new Main();
12     Beregner b = new Beregner();
13     static String CPR;
14
15     @FXML
16     TextField Username;
17     @FXML
18     PasswordField Password;
19 }
```

Figure 19: ControllerLogin variabler

Her ser vi importering af biblioteker, initialisere variabler og konstruktion objekter. Vi bruger også @FXML til at tilgå værdierne i tekst felter. Koden kan ses i figur: 19.

6.2.2 Metoder

```
20 public void login() throws IOException {
21     if (Kontroll()) {
22         // Læge skal kunne tilgå det hele
23         m.openStage(filename: "ProgramChooser.fxml", m.stage);
24     } else if (KontrolSP()) {
25         // sundhedspersonale skal kunne tilgå det meste, undtagen alarmgrænser.
26         m.openStage(filename: "ProgramChooserNoAlarm.fxml", m.stage);
27     } else if (KontrolP()) {
28         // patienter skal kun kunne tilgå deres arkiv.
29         m.openStage(filename: "PatientArkiv.fxml", m.stage);
30     }
31     else {
32         b.error("Forkert adgangskode");
33     }
34 }
```

Figure 20: ControllerLogin login()

Her kan vi se vores login metode, som kalder Kontroll, KontrolSP og KontrolP. Disse metoder returnere en boolsk værdi, og kontrolstrukturen vælger så en stage at vise. Det er på denne måde, vi adskiller læger, sundhedspersonale og patienter. Kontrolstrukturen kontrollerer efter vigtighed, dvs. først spørger den om det er en læge, dernæst sundhedspersonale og til sidst patient. Når vi skal åbne en stage, bruger vi openstage metoden fra main klassen. Hvis det indtastede brugernavn ikke eksistere i vores databaser, vil der kaldes en error metode fra vores beregner klasse. Databasen forklares i de følgende metoder. Koden kan ses i figur: 20.

```

36 private boolean kontrolP() {
37     //Hvis dit CPR findes at PatientData folderen, kan du logge ind
38     String U = Username.getText();
39     String P = Password.getText();
40     if (U != "") {
41         File checker = new File( parent: "PatientData", U);
42         CPR = U;
43         if (checker.exists()) {
44             return true;
45         }
46     }
47     return false;
48 }

```

Figure 21: ControllerLogin kontrolP()

KontrolP metoden står for Kontrol Patient. Den undersøger, om der eksisterer en fil med et navn tilsvarende til det indtastede brugernavn. Vi laver et file objekt, og bruger den tilhørende. Exists metoden bliver brugt til at returnere en boolsk værdi. Koden kan ses i figur: 21.

```

49 private boolean kontrolL() {
50     // her skal SQL implementeres
51     String[] Uarkiv = new String[3];
52     String[] Parkiv = new String[3];
53     Uarkiv[0] = "DR";
54     Parkiv[0] = "Password";
55     String U = Username.getText();
56     String P = Password.getText();
57
58     for (int i = 0; i < Uarkiv.length; i++) {
59         if (U.equals(Uarkiv[i]) && P.equals(Parkiv[i])) {
60             return true;
61         }
62     }
63     return false;
64 }

```

Figure 22: ControllerLogin kontrolL()

KontrollL metoden sammenligner det indtastede brugernavn og kodeord med værdierne i nogle String array. Det gør vi ved hjælp af for-loop kontrolstrukturen. Denne metode er lavet med intentionen, at der netop her, skal implementeres en SQL database. Herved har vi har forsøgt at lave en struktur, som imiterede funktionaliteten af dette. Koden kan ses i figur: 22.

```
74 private boolean kontrolSP() {  
75     // her skal SQL implementeres  
76     String[] Uarkiv = new String[3];  
77     String[] Parkiv = new String[3];  
78     Uarkiv[0] = "nurse";  
79     Parkiv[0] = "Password";  
80     String U = Username.getText();  
81     String P = Password.getText();  
82  
83     for (int i = 0; i < Uarkiv.length; i++) {  
84         if (U.equals(Uarkiv[i]) && P.equals(Parkiv[i])) {  
85             return true;  
86         }  
87     }  
88     return false;  
89 }  
90 }
```

Figure 23: ControllerLogin kontrolSP()

KontrolSP metoden fungerer på samme måde som KontrollL metoden. Den sammenligner de indtastede værdier med værdierne fra et arkiv, og returnere på baggrund af dette, en boolsk værdi. Koden kan ses i figur: 23.

6.3 ControllerProgramChooser

```
1 package sample;
2
3 import java.io.IOException;
4
5 public class ControllerProgramChooser {
6     Main m = new Main();
7     //alle metoder laver ny stage med FXML som scene
8     public void pulsTempClicked() throws IOException {
9         m.openStage( (filename: "Puls_SpO2_Temp.fxml");
10     }
11
12
13     public void arkivClicked() throws IOException {
14         m.openStage( (filename: "Arkiv.fxml");
15     }
16
17     public void ekgClicked() throws IOException {
18         m.openStage( (filename: "EKG_%FXML");
19     }
20
21     public void alarmClicked() throws IOException {
22         m.openStage( (filename: "Alarm.fxml");
23     }
24 }
```

Figure 24: ControllerProgramChooser

ControllerProgramChoser klassen tilhører den stage, som bliver åbnet fra login klassen med ControllerProgramChoser.fxml, som er den tilhørende controller til denne fxml fil. I Klassen er der fire metoder, der er tilsvarende til de fire knapper, som befinder sig på denne stage. Hver knap åbner en ny stage. Denne stage er designet, som værende den til læger, dvs. alt funktionalitet kan tilgås. Koden kan ses i figur: 24.

6.4 ControllerProgramChooserNoAlarm

```
1 package sample;
2
3 import java.io.IOException;
4
5 public class ControllerProgramChooserNoAlarm extends ControllerProgramChooser {
6     //Alle metoder laver ny stage med nye scene, denne klasse indeholder ikke en alarm knap
7     public void pulsTemp2() throws IOException {
8         pulsTempClicked();
9     }
10
11     public void arkiv2() throws IOException {
12         arkivClicked();
13     }
14
15     public void ekgClicked2() throws IOException {
16         ekgClicked();
17     }
18 }
```

Figure 25: ControllerProgramChooserNoAlarm

ControllerProgramChoserNoAlarm klassen tilhører den stage, som bliver åbnet fra login klassen, med ControllerProgramChoserNoAlarm.fxml. Den fungerer altså som ControllerProgramChoser klassen, og af den grund ekstender den også denne. På samme måde har den metoder tilsvarende til knapper. Dog har den ikke adgang til alarm.fxml stage'en, fordi denne er ment til sundhedspersonalet, som ikke skal kunne ændre alarmgrænser. Koden kan ses i figur: 25.

6.5 FileHandler

6.5.1 Variabler

```
1 package sample;
2
3 import javafx.embed.swing.SwingFXUtils;
4 import javafx.scene.snapshot.Parameters;
5 import javafx.scene.chart.LineChart;
6 import javafx.scene.image.WritableImage;
7
8 import javax.imageio.ImageIO;
9 import java.io.File;
10 import java.io.PrintWriter;
11 import java.io.IOException;
12
13 public class FileHandler {
14     String path;
15     PrintWriter fl;
16     String cpr;
```

Figure 26: FileHandler variabler

I klassen FileHandler har vi kun En FileWriter variable til et filewriter objekt, samt en String der hedder Path. Path bliver brugt, når vi skal finde vej til der, hvor vores directory skal laves, og til hvor vores data skal gemmes. Koden kan ses i figur: 26.

6.5.2 Metoder

```
11 //Laver en directory i konstruktøren
12 public FileHandler(String cpr) {
13     this.path = "PatientData/" + cpr;
14     System.out.println(path);
15     File folder = new File(path);
16     folder.mkdir();
17 }
```

Figure 27: FileHandler Constructor

I konstruktøren sættes path til placeringen af vores directory, som laves via mkdir() metoden. Mkdir() fungerer således, hvis der allerede er lavet en directory smider den en fejl, og går videre. Denne fejl kan man så senere smide bort. Koden kan ses i figur: 27.

```

27 //Save date gemmer en int
28 public void saveData(String type, String bogstav, int value) throws IOException {
29     f1 = new FileWriter((this.path + "\\\" + type), append: true);
30     f1.write(str: bogstav + "," + value + "," + "null" + "|");
31     f1.flush();
32 }
33
34 //Save date double gemmer en double
35 public void saveDataDouble(String type, String bogstav, double value) throws IOException {
36     f1 = new FileWriter((this.path + "\\\" + type), append: true);
37     f1.write(str: bogstav + "," + value + "," + "null" + "|");
38     f1.flush();
39 }

```

Figure 28: FileHandler saveData() og saveDataDouble()

SaveData metoderne fungerer således, at der skrives til en fil, som man selv bestemmer navnet på via type. SaveData skriver i følgende syntaks: Tid,Værdi,Kommentar—Tid,Værdi,Kommentar— osv. Kommentarfeltet bruges henholdsvis ikke til noget, men er noget der arbejdes på. Koden kan ses i figur: 28.

```

41 @ public void saveAsPng(LineChart lineChart,String name) {
42     File file1 = new File( pathname: "journal Billeder/" + cpr);
43     file1.mkdir();
44     WritableImage image = lineChart.snapshot(new SnapshotParameters(), writableImage: null);
45     File file = new File( pathname: "journal Billeder/" + cpr + "/" + name);
46     try {
47         ImageIO.write(SwingFXUtils.fromFXImage(image, bufferedImage: null), formatName: "png", file);
48     } catch (IOException e) {
49         e.printStackTrace();
50     }
51 }
52 }

```

Figure 29: FileHandler saveAsPng()

saveAsPng metoden fungerer således, at den først generere en directory under Journal map- pen. Denne metode bliver kørt 4 gange. 1 for hver linechart. mkdir() metoden generere kun en mappe. Dernæst generere den en png fil af linechart, og gemmer den. Koden kan ses i figur: 29.

6.6 Beregner

6.6.1 Variabler

```
1 package sample;
2
3 import java.util.*;
4
5 public class Beregner {
6     Main m = new Main();
7
8     //Variabler til puls, temperatur,ekg simulation og fremvisning
9     double nametest, math, math2, math3, temp, SpO2double;
10    int puls, red, u, pulseCheck, tempCheck, i;
11    String SpO2, SpO2String;
12    double intervalmin = 70;
13    double intervalmax = 70;
14    double intervalmin2 = 98;
15    double intervalmax2 = 100;
16    double intervalmin3 = 36;
17    double intervalmax3 = 38;
18
19    //Værdier brugt til Alarmgrænser
20    static double pulseMaxDouble = 220;
21    static double pulseMinDouble = 0;
22    static double tempMaxDouble = 49;
23    static double tempMinDouble = 35;
24    static double SpO2MaxDouble = 100;
25    static double SpO2MinDouble = 95;
26    static double ekgMaxDouble = 150;
27    static double ekgMinDouble = -50;
28    int alarmCounter=-50;
29
30    //String brugt til holde CPR gemt, mellem stages.
31    static String name;
32
33    //Objekter brugt til at fremvise linechart serie 1 realtid
34    ScheduledExecutorService Eventhandler = Executors.newSingleThreadScheduledExecutor();
35    XYChart.Series pulsSeries = new XYChart.Series();
36    XYChart.Series temperatureSeries = new XYChart.Series();
37}
```

Figure 30: FileHandler Beregner variabler

I Beregnerklassen findes variabler, som bruges til optælling og simulering. Dernæst lagres alarmgrænserne i globale variabler, samt det indtastede CPR bliver gemt mellem PulsSpO2Temp og EKG. Til slut dannes Objekter til fremvisning af data på linecharts, og en ScheduledExecutorService til at fremvise dataen i realtid. Koden kan ses i figur: 30.

6.6.2 Metoder

I monitorStartPuls() testes der først, om det indtastede CPR kan laves til en double. Dernæst testes om CPR er 10 cifre lang. Så simuleres der data, og bliver vist på linechartet via. Eventhandleren, som her er sat til 1 sekund mellem hver tråd, hvorefter den gemmer dataen i en fil. På scenen ses 2 radiobuttons, som styrer hvilke data, der skal blive vist på linecharten. Hvis den ikke kan dette, kalder den error(). Koden kan ses i figur: 31.

```

58 //metode til at fremvise puls,temp og spo2
59 public void monitorStartPuls(TextField textField, LineChart<CategoryAxis, NumberAxis> linechart,
60                             Label label, Label label2) throws IOException {
61     this.name = textField.getText();
62     try {
63         nametest = Double.parseDouble(name);
64         if (name.length() == 10) {
65             FileHandler fh = new FileHandler(name);
66             pulsSeries.setName("puls");
67             temperatureSeries.setName("Temperature");
68             linechart.getData().addAll(pulsSeries, temperatureSeries);
69             Eventhandler.scheduleAtFixedRate(() -> {
70                 Platform.runLater(() -> {
71                     String bogstav = String.valueOf(i);
72                     SpO2Simulation();
73                     SpO2String += bogstav + " " + SpO2 + " ";
74                     pulseSimulation();
75                     temperatureSimulation();
76                     if (tempCheck == 1) {
77                         label2.setText((Temp + "°C"));
78                         temperatureSeries.getData().add(new XYChart.Data(bogstav, temp));
79                         alarmCheck(string: "ALARM TEMPERATUR ER FARLIG", tempMaxDouble, tempMinDouble, temp,1);
80                     }
81                     try {
82                         fh.saveDataDouble(type: "Temp", bogstav, temp);
83                     } catch (IOException e) {
84                         e.printStackTrace();
85                     }
86                     if (pulseCheck == 1) {
87                         pulsSeries.getData().add(new XYChart.Data(bogstav, puls));
88                         alarmCheck(string: "ALARM PULS ER FARLIG", pulseMaxDouble, pulseMinDouble, puls,1);
89                     }
90                     try {
91                         fh.saveData(type: "Pulse", bogstav, puls);
92                     } catch (IOException e) {
93                         e.printStackTrace();
94                     }
95                     if (i % 2 == 0) {
96                         label.setText(SpO2);
97                         alarmCheck(string: "SP02 ER FARLIG", SpO2MaxDouble, SpO2MinDouble, SpO2double,1);
98                     }
99                     try {
100                         fh.saveDataDouble(type: "SpO2", bogstav, SpO2double);
101                     } catch (IOException e) {
102                         e.printStackTrace();
103                     }
104                 }
105             }, i++);
106             (initialDelay: 0, period: 1, TimeUnit.SECONDS);
107         } else {
108             error("Invalid input, Cpr skal være 10 cifre");
109         }
110     } catch (Exception e) {
111         error("Invalid input, CPR skal være tal");
112     }
113 }
114 }
115

```

Figure 31: Beregner monitorStartPuls()

```

113 //Metode til at stoppe fremvisning i realtid af puls, temp og spo2
114 public void monitorStopPuls() throws IOException {
115     Eventhandler.shutdown();
116 }

```

Figure 32: Beregner monitorStopPuls()

monitorStopPuls() stopper bare eventhandleren. Dette gør dog, at metoden monitorStartPuls() ikke kan køre igen, uden at man lukker stagen, og åbner den igen. Dette vurderede vi til, at tage for lang tid at fikse, i forhold til hvad det gjorde for funktionaliteten. Koden kan ses i figur: 32.

```

118 //Kontrol logic in radiobutton pulse
119 public int showPulsePuls() {
120     if (pulseCheck == 1) {
121         pulseCheck = 0;
122         return pulseCheck;
123     }
124     if (pulseCheck == 0) {
125         pulseCheck = 1;
126         return pulseCheck;
127     }
128     return 0;
129 }

```

Figure 33: Beregner showPulsePuls()

showPulsePuls() er en metode, der findes på radiobuttons, der returnere 1, hvis knappen er trykket ind, og 0 når knappen bliver trykket ud. Denne burde have været en boolean eller void, og bliver kaldt i koden, i stedet for at der bliver kaldt variabelen pulsecheck. Koden kan ses i figur: 33.

```

131 //Kontrol logic in radiobutton temp
132 public int showTemperaturePuls() {
133     if (tempCheck == 1) {
134         tempCheck = 0;
135         return tempCheck;
136     }
137     if (tempCheck == 0) {
138         tempCheck = 1;
139         return tempCheck;
140     }
141     return 0;
142 }

```

Figure 34: Beregner showTemperaturePuls()

showTemperaturePuls() er en metode, der findes på radiobuttons, der returnere 1, hvis knappen er trykket ind, og 0 når knappen bliver trykket ud. Denne burde have været en boolean eller void, og bliver kaldt i koden, i stedet for at der bliver kaldt variabelen tempcheck. Koden kan ses i figur: 34.

```

147 //metode til at kontrollere alarm
148 public void alarmCheck(String string, double alarmMax, double alarmMin, double value,int tæller) {
149     if (value < alarmMin || value > alarmMax && (tæller-alarmCounter)>50) {
150         error(string);
151         alarmCounter=tæller;
152     }
153 }

```

Figure 35: Beregner alarmCheck()

alarmCheck() kontrollere om alarMin er midre end alarmMax, hvis dette er fakskt falskt, fremvises error() metoden. Koden kan ses i figur: 35.

```

155 public void pulseSimulation() {
156     math = Math.random() * (intervalmax - intervalmin) + intervalmin;
157     puls = (int) math;
158     intervalmax = math + 5;
159     intervalmin = math - 5;
160 }
161
162 public void SpO2Simulation() {
163     math2 = Math.random() * (intervalmax2 - intervalmin2) + intervalmin2;
164     SpO2double = new BigDecimal(math2).setScale( newScale: 2, RoundingMode.HALF_UP).doubleValue();//Runder til 2 decimal
165     Spo2 = (SpO2double + "%");
166     intervalmax2 = math2 + 0.05;
167     intervalmin2 = math2 - 0.05;
168 }
169
170 public void temperatureSimulation() {
171     math3 = Math.random() * (intervalmax3 - intervalmin3) + intervalmin3;
172     temp = new BigDecimal(math3).setScale( newScale: 2, RoundingMode.HALF_UP).doubleValue(); //Runder til 2 decimal
173     intervalmax3 = math3 + 0.25;
174     intervalmin3 = math3 - 0.25;
175 }
176
177 //EKG værdier array
178 double Red[] = {0, 10, 15, 20, 15, 10, 0, 0, -10, 100, -30, 0, 0, 5, 10, 20, 25, 30, 20, 10, 5, 0};
179
180 public void ekgSimulation() {
181     if (u < 21) {
182         u++;
183     } else {
184         u = 0;
185     }
186     red = (int) Red[u];
187 }
188
189 public int redv() { return red; }
190
191
192

```

Figure 36: Beregner Simuleringer

Her simuleres dummy data i figur: 36.


```

155     public void pulseSimulation() {
156         math = Math.random() * (intervalmax - intervalmin) + intervalmin;
157         puls = (int) math;
158         intervalmax = math + 5;
159         intervalmin = math - 5;
160     }
161
162     public void SpO2Simulation() {
163         math2 = Math.random() * (intervalmax2 - intervalmin2) + intervalmin2;
164         SpO2double = new BigDecimal(math2).setScale(newScale, RoundingMode.HALF_UP).doubleValue(); //Runder til 2 decimal
165         SpO2 = (SpO2double + "%");
166         intervalmax2 = math2 + 0.05;
167         intervalmin2 = math2 - 0.05;
168     }
169
170     public void temperatureSimulation() {
171         math3 = Math.random() * (intervalmax3 - intervalmin3) + intervalmin3;
172         temp = new BigDecimal(math3).setScale(newScale, RoundingMode.HALF_UP).doubleValue(); //Runder til 2 decimal
173         intervalmax3 = math3 + 0.25;
174         intervalmin3 = math3 - 0.25;
175     }
176
177     //EKG værdier array
178     double Red[] = {0, 10, 15, 20, 15, 10, 0, 0, -10, 100, -30, 0, 0, 5, 10, 20, 25, 30, 20, 10, 5, 0};
179
180     public void ekgsimulation() {
181         if (u < 21) {
182             u++;
183         } else {
184             u = 0;
185         }
186         red = (int) Red[u];
187     }
188
189     public int redv() { return red; }
190
191

```

Figure 37: Beregner error()

error() metoden åbner en stage med en label og en knap, således at man bliver promptet med en besked, hvorefter man skal confirme med OK for at komme videre. Koden kan ses i figur: 37.

6.7 ControllerAlarm

6.7.1 Variabler

```
1 package sample;
2
3 import ...
4
5
6
7
8
9 public class ControllerAlarm extends Beregner implements Initializable {
10
11     //Finder textfields i fxml
12     public TextField pulseMax;
13     public TextField pulseMin;
14     public TextField tempMax;
15     public TextField tempMin;
16     public TextField SpO2Max;
17     public TextField SpO2Min;
18     public TextField ekgMax;
19     public TextField ekgMin;
20     private int a = 0;
21 }
```

Figure 38: ControllerAlarm controllerAlarm variabler

ControllerAlarm klassen indeholder TextFields og en variabel til kontrol. Koden kan ses i figur: 38.

6.7.2 Metoder

```
22 public void updateAlarm() {
23     a = 0;
24     check(pulseMax, pulseMin);
25     check(tempMax, tempMin);
26     check(SpO2Max, SpO2Min);
27     check(ekgMax, ekgMin);
28
29     if (a == 4) {
30         getText();
31         setText();
32         error("Values updated");
33     }
34 }
```

Figure 39: ControllerAlarm updateAlarm()

Update alarm kontrollerer om Max er større end Min og henter teksten fra TextField, og opdatere deres plades i beregneren hvis alle typer er i ordentlig syntaks. Koden kan ses i figur: 39.

```

36 @ public void check(TextField text, TextField text2) {
37     try {
38         double max = Double.parseDouble(text.getText());
39         double min = Double.parseDouble(text2.getText());
40         if (max > min) {
41             a++;
42         } else {
43             error("Max has to be greater than Min");
44         }
45     } catch (NumberFormatException e) {
46         error("Invalid Number, Write a real number comma with . ");
47     }
48 }

```

Figure 40: ControllerAlarm check()

Check() kontrollerer om max er større end min, og tæller tælleren op. Koden kan ses i figur: 40.

```

51 public void setText() {
52     pulseMax.setText(String.valueOf(pulseMaxDouble));
53     pulseMin.setText(String.valueOf(pulseMinDouble));
54     tempMax.setText(String.valueOf(tempMaxDouble));
55     tempMin.setText(String.valueOf(tempMinDouble));
56     SpO2Max.setText(String.valueOf(SpO2MaxDouble));
57     SpO2Min.setText(String.valueOf(SpO2MinDouble));
58     ekgMax.setText(String.valueOf(ekgMaxDouble));
59     ekgMin.setText(String.valueOf(ekgMinDouble));
60 }
61
62 public void getText() {
63     pulseMaxDouble = Double.parseDouble(pulseMax.getText());
64     pulseMinDouble = Double.parseDouble(pulseMin.getText());
65     tempMaxDouble = Double.parseDouble(tempMax.getText());
66     tempMinDouble = Double.parseDouble(tempMin.getText());
67     SpO2MaxDouble = Double.parseDouble(SpO2Max.getText());
68     SpO2MinDouble = Double.parseDouble(SpO2Min.getText());
69     ekgMaxDouble = Double.parseDouble(ekgMax.getText());
70     ekgMinDouble = Double.parseDouble(ekgMin.getText());
71 }

```

Figure 41: ControllerAlarm setText() og getText()

setText() og getText() sætter værdien i beregneren og henter værdien fra textfields. Koden kan ses i figur: 41.

```

74 @Override
75 public void initialize(URL url, ResourceBundle resourceBundle) { setText(); }
76

```

Figure 42: ControllerAlarm Initialize()

Initialize() metoden kører når klassen åbnes og sætter teksten på TextFields, så den er overens med værdierne i beregner. Koden kan ses i figur: 42.

closeScene() metoden kalder metoden i Main der hedder closeStage. Koden kan ses i figur: 43.

```

77 public void closeScene(ActionEvent actionEvent) {
78     m.closeStage(n.stage2);
79 }
80 }

```

Figure 43: ControllerAlarm closeScene()

6.8 ControllerArkiv

6.8.1 Variabler

```

1 package sample;
2
3 import javafx.event.ActionEvent;
4 import javafx.fxml.*;
5 import javafx.scene.chart.LineChart;
6 import javafx.scene.chart.NumberAxis;
7 import javafx.scene.chart.XYChart;
8 import javafx.scene.control.TextField;
9
10 import java.util.*;
11 import java.io.*;
12
13 public class ControllerArkiv extends Beregner {
14     @FXML
15     public TextField timeMin;
16     @FXML
17     public TextField timeMax;
18     @FXML
19     public NumberAxis spO2AKse;
20     @FXML
21     public NumberAxis EKGAKse;
22     @FXML
23     public NumberAxis tempAKse;
24     @FXML
25     public NumberAxis pulseAKse;
26     @FXML
27     TextField CPR;
28
29     @FXML
30     LineChart<NumberAxis, NumberAxis> PulseChart;
31     @FXML
32     LineChart<NumberAxis, NumberAxis> TempChart;
33     @FXML
34     LineChart<NumberAxis, NumberAxis> SpO2Chart;
35     @FXML
36     LineChart<NumberAxis, NumberAxis> EKGChart;
37
38     XYChart.Series pulseXYChart = new XYChart.Series();
39     XYChart.Series tempXYChart = new XYChart.Series();
40     XYChart.Series spO2XYChart = new XYChart.Series();
41     XYChart.Series EKGXYChart = new XYChart.Series();
42
43     int[] pulseTime, tempTime, spO2Time, EKGTime;
44     double[] pulseValue, tempValue, spO2Value, EKGValue;
45     String[] pulseArray, tempArray, spO2Array, EKGArray;
46     int timeMaxInt = 60;
47     int timeMinInt = 0;

```

Figure 44: ControllerArkiv variabler

I klassen controllerArkiv deklareres variabler til indsættelse af data på linechart, og opdeling af data fra filerne. Der tilgås også linacharts samt tekstfelter med @FXML, desuden konstrueres XYChart objekter. Koden kan ses i figur: 44.

6.8.2 Metoder

```

49 public void PatientChooser() throws FileNotFoundException {
50     if (cprCheck2()) {
51         error("CPR-nummer er godkendt");
52     } else {
53         error("Ugyldigt CPR-nummer");
54     }
55 }

```

Figure 45: ControllerArkiv PatientChooser()

PatientChooser() sørge for at give error beskeder hvis CPR er forkert eller ugyldigt. Den gør dette ved at kalde cprCheck2 metoden som returnere en boolsk værdi. Koden kan ses i figur: 45.

```

57 public void PulsArkiv() throws FileNotFoundException {
58     populateChart( filetype: "Pulse", pulsArray, PulseXYChart, PulseChart, PulseTime, PulseValue, pulseXakse);
59 }
60
61 public void TempArkiv() throws FileNotFoundException {
62     populateChart( filetype: "Temp", tempArray, TempXYChart, TempChart, TempTime, TempValue, tempXakse);
63 }
64
65 public void SpO2Arkiv() throws FileNotFoundException {
66     populateChart( filetype: "SpO2", SpO2Array, SpO2XYChart, SpO2Chart, SpO2Time, SpO2Value, SpO2Xakse);
67 }
68
69 public void EKGArkiv() throws FileNotFoundException {
70     populateChart( filetype: "EKG", EKGArray, EKGXYChart, EKGChart, EKGTime, EKGValue, EKGXakse);
71 }
72 }
73
74 public String CPR() { return CPR.getText(); }
75
76
77
78

```

Figure 46: ControllerArkiv PulsArkiv(), TempArkiv(), SpO2Arkiv(), EKGArkiv() og CPR()

PulsArkiv(), TempArkiv(), SpO2Arkiv(), EKGArkiv(), er metoder der kaldes på knapperne i arkivet. CPR() bruges til at sætte det rigtige CPR nr. ved at tilgå værdien af tekstfeltet CPR. Koden kan ses i figur: 46.

```

79 //metode til at fange data og indlæse det i scanner.
80 public void populateChart(String filetype, String[] array, XYChart.Series xyChart, LineChart lineChart,
81     int[] time, double[] value, NumberAxis xakse) throws FileNotFoundException {
82
83     if (cprCheck2()) {
84         xyChart.getData().clear();
85         lineChart.getData().clear();
86
87         String FileName = CPR();
88         File Pulse1 = new File( pathname: "PatientData/" + FileName + "/" + filetype); //mac :FileName, "Pulse"
89         Scanner Patient = new Scanner(Pulse1);
90         String PulseData = Patient.nextLine();
91
92         String RÅ = PulseData.replaceAll( regex: "[^0-9.]", (replacement: ""));
93         array = RÅ.split( regex: "[.]" );
94
95         time = new int[array.length / 2];
96         if (array.length > 1) {
97             for (int i = 0; i < array.length; i = i + 2) {
98                 time[i / 2] = Integer.parseInt(array[i]);
99             }
100         }
101
102         value = new double[array.length / 2];
103         if (array.length > 1) {
104             for (int i = 1; i < array.length; i = i + 2) {
105                 value[i / 2] = Double.parseDouble(array[i]); // hvad sker der når man deler 3 med 2 som integer.
106             }
107         }
108
109         if (timeMax.getText() != "null" || timeMin.getText() != "null") {
110             try {
111                 timeMaxInt = Integer.parseInt(timeMax.getText());
112                 timeMinInt = Integer.parseInt(timeMin.getText());
113             } catch (NumberFormatException e) {
114                 timeMaxInt = value.length;
115                 timeMax.setText(String.valueOf(timeMaxInt));
116                 timeMinInt = 0;
117                 timeMin.setText(String.valueOf(timeMinInt));
118             }
119         }
120
121         if (timeMaxInt > value.length) {
122             timeMaxInt = value.length;
123             timeMax.setText(String.valueOf(timeMaxInt));
124         }
125
126         for (int a = 0; a < timeMaxInt; a++) {
127             xyChart.getData().add(new XYChart.Data(time[a], value[a]));
128         }
129         xakse.setUpperBound(timeMaxInt);
130         xakse.setLowerBound(timeMinInt);
131         lineChart.getData().add(xyChart);
132     } else {
133         error("Ugyldigt cpr");
134     }
135 }
136

```

Figure 47: ControllerArkiv populateChart()

populateChart() finder den rigtige fil, opdeler dataen i arrays, kontrollere tidsintervaller man vil vise, og skriver daten til linechartet. Koden kan ses i figur: 47.

```

125 public boolean cprCheck2(){
126     // metode som kontrollere om der er indtastet et eksisterende cpr
127     File checker = new File( parent: "PatientData", CPR());
128     if (checker.exists() && CPR().length() > 0) {
129         return true;
130     } else {
131         return false;
132     }
133 }
134 }

```

Figure 48: ControllerArkiv cprCheck2()

cprCheck2() tester om filen eksistere og længden på CPR-nummeret er 10. Koden kan ses i figur: 48.

```

147 public void saveDataToJournal(ActionEvent actionEvent) {
148     FileHandler fh= new FileHandler(CPR.getText());
149     fh.saveAsPng(PulseChart, name: "Puls.png");
150     fh.saveAsPng(TempChart, name: "Temp.png");
151     fh.saveAsPng(EKGChart, name: "EKG.png");
152     fh.saveAsPng(SpO2Chart, name: "SpO2.png");
153 }
154 }
155
156 public void closeScene(ActionEvent actionEvent) {
157     m.closeStage(m.stage2);
158 }
159 }

```

Figure 49: ControllerArkiv saveDataToJournal() og closeScene()

closeScene() metoden kalder metoden i Main der hedder closeStage, hvor saveDataToJournal gemmer linecharts i PNG filer. Koden kan ses i figur: 49.

6.9 ControllerPatientArkiv

6.9.1 Variabler

```
1 package sample;
2
3 import javafx.fxml.FXML;
4 import javafx.fxml.Initializable;
5 import javafx.scene.chart.LineChart;
6 import javafx.scene.chart.NumberAxis;
7 import javafx.scene.control.Label;
8 import javafx.scene.control.TextField;
9 import java.io.FileNotFoundException;
10 import java.net.URL;
11 import java.util.ResourceBundle;
12
13 public class ControllerPatientArkiv extends ControllerArkiv implements Initializable {
14
15     ControllerLogin CL = new ControllerLogin();
16
17     @FXML
18     public TextField timeMin;
19     @FXML
20     public TextField timeMax;
21     @FXML
22     public Label Cprlabel;
23     @FXML
24     public NumberAxis EKGxAKse2;
25     @FXML
26     public NumberAxis SpO2xAKse2;
27     @FXML
28     public NumberAxis tempxAKse2;
29     @FXML
30     public NumberAxis pulsexAKse2;
31     @FXML
32     LineChart<NumberAxis, NumberAxis> PulseChart;
33     @FXML
34     LineChart<NumberAxis, NumberAxis> TempChart;
35     @FXML
36     LineChart<NumberAxis, NumberAxis> SpO2Chart;
37     @FXML
38     LineChart<NumberAxis, NumberAxis> EKGChart;
39 }
```

Figure 50: ControllerPatientArkiv variabler

ControllerPatientArkiv indeholder variabler til dets linecharts og scenebuilder elementer. Koden kan ses i figur: 6.9.

6.9.2 Metoder

```
33 public void PulsArkiv() throws FileNotFoundException {
34     populateChart( filetype: "Pulse", pulsArray, PulseXYChart, PulseChart, PulseTime, PulseValue);
35 }
36
37 public void TempArkiv() throws FileNotFoundException {
38     populateChart( filetype: "Temp", tempArray, TempXYChart, TempChart, TempTime, TempValue);
39 }
40
41 public void SpO2Arkiv() throws FileNotFoundException {
42     populateChart( filetype: "SpO2", SpO2Array, SpO2XYChart, SpO2Chart, SpO2Time, SpO2Value);
43 }
44
45 public void EKGArkiv() throws FileNotFoundException {
46     populateChart( filetype: "EKG", EKGArray, EKGXYChart, EKGChart, EKGTime, EKGValue);
47 }
48
49 @Override
50 public String CPR() { return Cprlabel.getText(); } //En patient skal kun kunne tilgå sine egne data
```

Figure 51: ControllerPatientArkiv, TempAkriv(), SpO2Arkiv(), EKGArkiv() og CPR()

PulsArkiv(), TempArkiv(), SpO2Arkiv() og EKGArkiv() bruger alle metoden populateChart fra ControllerArkiv. CPR() metoden bliver skiftet ud til returnere værdien fra CprLabel, da man som patient ikke skal kunne tilgå de andres værdier og helst skal undgå at skrive sit CPR nr. 2 gange. Koden kan ses i figur: ??.

```

54  @Override //En patient skal kun kunne tilgå sine egne data, derfor bliver CPR automatisk overført
55  public void initialize(URL url, ResourceBundle resourceBundle)
56  {
57      Cprlabel.setText(CL.CPR);
58      // populære charts fra start
59      try {
60          PulsArkiv();
61          TempArkiv();
62          SpO2Arkiv();
63          EKGArkiv();
64      } catch (FileNotFoundException e) {
65          e.printStackTrace();
66      }
67  }
68  }
69  }

```

Figure 52: ControllerPatientArkiv Initialize()

Initialize() sætter CprLabel til at indholde værdien som man skrev i controlLogin, så man ikke skal skrive den 2 gange, samtidig med at den udfylder linecharts automatisk. Koden kan ses i figur: 52.

6.10 EKGController

6.10.1 Variabler

```
1 package sample;
2
3 import javafx.application.Platform;
4 import javafx.fxml.*;
5 import javafx.scene.chart.LineChart;
6 import javafx.scene.chart.XYChart;
7 import javafx.scene.control.TextField;
8 import java.io.IOException;
9 import java.net.URL;
10 import java.util.ResourceBundle;
11 import java.util.concurrent.TimeUnit;
12
13
14 public class EKGController extends Beregner implements Initializable {
15     static String navn;
16     double tjeK;
17     int y = 0;
18
19     @FXML
20     LineChart<String, Number> ekgplot;
21     @FXML
22     TextField CPRLabel;
23     //Data serie
24     XYChart.Series<String, Number> data = new XYChart.Series<String, Number>();
25 }
```

Figure 53: EKG variabler

EKGController indeholder variabler til optælling og kontrol af navn, samt dets elementer til fremvisning af data, på dets linechart. Koden kan ses i figur: 53.

6.10.2 Metoder

```
26 public void startEKG() {
27     y = 0;
28     navn = CPRLabel.getText();
29     try {
30         tjeck = Double.parseDouble(navn);
31         if (navn.length() == 10) {
32             name = navn;
33             FileHandler FL = new FileHandler(navn);
34             EventHandler.scheduleAtFixedRate(() ->
35                 Platform.runLater(() -> {
36                     ekgplot.getData().clear();
37                     ekgSimulation();
38                     String n = String.valueOf(y);
39                     int redval = redv();
40                     data.getData().add(new XYChart.Data<String, Number>(n, redval));
41                     ekgplot.getData().add(data);
42                     alarmCheck(string: "EKG ER FARLIG", ekgMaxDouble, ekgMinDouble, redval, y);
43                     try {
44                         FL.saveData(type: "EKG", n, redval);
45                     } catch (IOException e) {
46                         e.printStackTrace();
47                     }
48                     y++;
49                 }, initialDelay: 0, period: 100, TimeUnit.MILLISECONDS);
50             } else {
51                 error("Invalid input, Cpr skal være 10 cifre");
52             }
53         } catch (Exception e) {
54             error("Invalid input, CPR skal være tal");
55         }
56     }
57 }
```

Figure 54: EKGController startEKG()

startEKG() metoden tester og det indskrevne CPR nr. er tal og præcis 10 cifre. Dernæst gemmer den værdien af CPR og kører Eventhandleren på 100 milisekunder mellem hver tråd, hvor den skriver dataen til linechart og gemmer dataen i en fil. Hvis den ikke kan dette kalder den error(). Koden kan ses i figur: 54.

```
58 public void EKGstop() {
59     EventHandler.shutdown();
60 }
61
62 @Override
63 public void initialize(URL url, ResourceBundle resourceBundle) {
64     CPRLabel.setText(name);
65 } //Sætter CPR navn
66 }
```

Figure 55: EKGController EKGstop() og Initialize()

EKGstop() stopper eventhandleren, og har dog samme problem som nævnt under Beregner/metoder/monitorStart(). Initialize(), sætter værdien af CPR, hvis den findes, på dets plads, således at man ikke skal skrive det samme CPR nr. 2 gange under undersøgelser. Koden kan ses i figur: 55.

closeScene() metoden kalder metoden i Main der hedder closeStage. Koden kan ses i figur: 56.

```

68     public void closeScene(ActionEvent actionEvent) {
69         if (EventHandler.isShutdown()==false){
70             EKGstop();
71         }
72         m.closeStage(m.stage2);
73     }
74 }

```

Figure 56: EKGController closeScene()

6.11 PulsSpO2TempController

6.11.1 Variabler

```

1  package sample;
2
3  import javafx.fxml.FXML;
4  import javafx.fxml.Initializable;
5  import javafx.scene.chart.*;
6  import javafx.scene.control.Label;
7  import javafx.scene.control.TextField;
8
9  import java.io.IOException;
10 import java.net.URL;
11 import java.util.ResourceBundle;
12
13 public class Puls_SpO2_TempController extends Beregner implements Initializable {
14
15     @FXML
16     Label tempLabel;
17     @FXML
18     LineChart<CategoryAxis, NumberAxis> Diagram;
19     @FXML
20     TextField Name;
21     @FXML
22     Label Spo2Label;
23 }

```

Figure 57: PulsSpO2TempController variabler

PulsSpO2TempController klassen indeholder kun variabler til sine scenebuilder elementer. Koden kan ses i figur: 57.

6.11.2 Metoder

```
25 public void monitorStart() throws IOException {
26     monitorStartPuls(Name, Diagram, Spo2Label, tempLabel);
27 }
28
29 public void monitorStop() throws IOException {
30     monitorStopPuls();
31 }
32
33 public void showPulse() { showPulsePuls(); }
34
35 public void showTemperature() { showTemperaturePuls(); }
36
37
38
39
40
41 @Override
42 public void initialize(URL url, ResourceBundle resourceBundle) {
43     Name.setText(name);
44     } //Bruger navnet fra en tidligere stage til at sætte CPR navn.
45
46 public void closeScene(ActionEvent actionEvent) {
47     if (EventHandler.isShutdown()==false){
48         EventHandler.shutdown();
49     }
50     m.closeStage(m.stage2);
51 }
52 }
```

Figure 58: PulsSpO2TempController metoder

Metoder brugt her findes i beregner klassen. Initialize(), sætter værdien af CPR, hvis den findes, på dets plads således at man ikke skal skrive det samme CPR nr. 2 gange under undersøgelser. closeScene() metoden kalder metoden i Main der hedder closeStage. Koden kan ses i figur: 58.

7 Afprøvning

I dette afsnit vil vi udfører en afprøvning af programmet på personaer, og afprøve vores programkode med White box og black box testing. Dette gøres for at få afprøvet programmet fra flere vinkler:

- Fra brugernes, set i afsnit 7.1
- Funktionel testing, herunder blackbox testing, i afsnit 7.4

7.1 Afprøvning på personaer

Under afprøvning af personaer, vil vi gennemgå vores programs funktionalitet og grafiske interface, med henblik på vores personaer. Vi vil prøve at undgå påvirkning af personaens holdning, ved eventuelle hjælpespørgsmål osv. Vi har lavet afprøvning på en Læge og en patient. Vi valgte en patient over en sygeplejerske, fordi at vi mente, at sygeplejersken ville give nogenlunde samme feedback som lægen.

7.1.1 Læge

Vi fik snakket med en læge, om hvordan vores program fungerede. Overordnet virkede det godt, og det var rart, at der var en simpel brugerflade og få antal knapper at klikke på. Der var dog også noget, der skulle gøres bedre eller nemmere at opdage. SpO2 og Temp Labels inde i PulsSpO2Temp klassen skulle være større, så de var mere iøjnefaldende, da de tog lidt lang tid at finde. Error boksene der dukkede op, når alarmgrænserne blev overskredet, skulle begrænses til en Error boks, der ikke kom op igen, når man klikkede ok. Inde i EKG klassen, manglede vi værdier på y akse. I Arkiv klassen var det godt, at man skulle skrive CPR igen, for at undgå at rapportere en forkert persons data i en anden journal. Tids-intervalknapperne skal være nemmere at se, og mere informative om hvad de gør. Vi spurgte om det eventuelt ville være rart, at man kunne sætte kommentarer til eventuelle hændelser, og fik som svar, at det ikke var ønsket, da det alligevel skulle dokumenteres i journalen. I stedet blev der ønsket en knap, så man kunne gemme billeder af diagrammerne visuelt i patientjournalen. Om det skete automatisk, eller at man selv skulle overføre dem var ligegyldigt, dog ville det selvfølgelig være rart, hvis programmet automatisk gjorde det.

7.1.2 Patient

Patienten forstod intuitivt, at når man havde været hos lægen eller på hospitalet, skulle man logge ind via sit CPR nr. Brugeren forstod også med det samme, at hvis man gerne ville se en eventuel bestemt hændelse, skulle man vælge tidsintervallerne i tekstfelterne.

7.2 White box

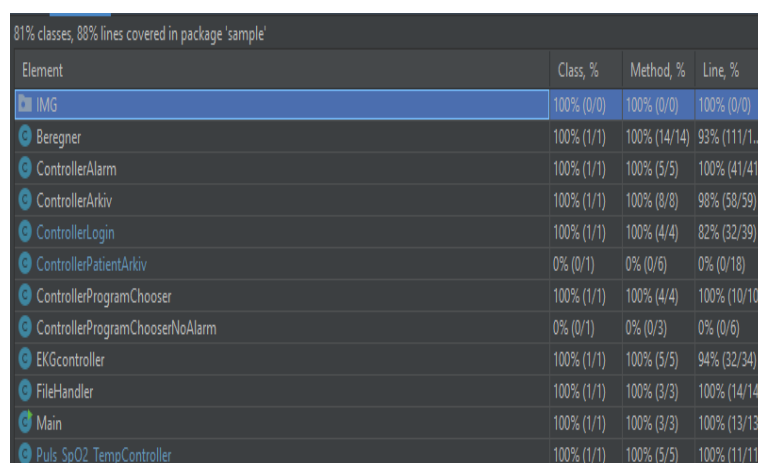
Under Whitebox testing bruger vi Basis path testing og Path testing.

7.2.1 Basis path testing

Basis path testing er en coverage på de enkelte scenerier beskrevet i aktivitetsdiagrammet. Målet her er, at se om programmet kører nogle metoder, som den ikke skal kører. Såvel som om den har adgang til klasser, den ikke skal have adgang til.

7.2.2 Læge

Der sprang ingen fejl frem, så vi udførte testen udendvidere. Derved kan man se, at sundhedspersonalet ikke har adgang til programChooserNoAlarm eller patientarkiv klassen som planlagt, samt alle dets planlagte metoder blev kørt. Dette kan ses på figur: ??.

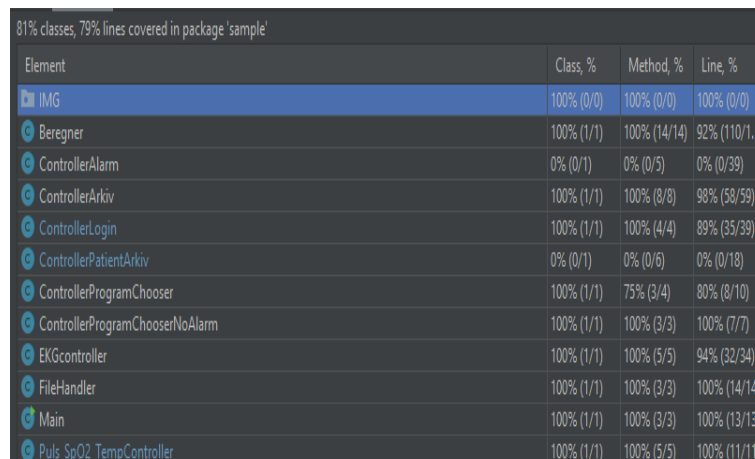


81% classes, 88% lines covered in package 'sample'			
Element	Class, %	Method, %	Line, %
IMG	100% (0/0)	100% (0/0)	100% (0/0)
Beregner	100% (1/1)	100% (14/14)	93% (111/1...
ControllerAlarm	100% (1/1)	100% (5/5)	100% (41/41)
ControllerArkiv	100% (1/1)	100% (8/8)	98% (58/59)
ControllerLogin	100% (1/1)	100% (4/4)	82% (32/39)
ControllerPatientArkiv	0% (0/1)	0% (0/6)	0% (0/18)
ControllerProgramChooser	100% (1/1)	100% (4/4)	100% (10/10)
ControllerProgramChooserNoAlarm	0% (0/1)	0% (0/3)	0% (0/6)
EKGcontroller	100% (1/1)	100% (5/5)	94% (32/34)
FileHandler	100% (1/1)	100% (3/3)	100% (14/14)
Main	100% (1/1)	100% (3/3)	100% (13/13)
Puls_SpO2_TempController	100% (1/1)	100% (5/5)	100% (11/11)

Figure 59: Basis path test læge

7.2.3 Sundhedspersonale

Der sprang ingen fejl frem, så vi kunne udføre testen. Derved kan man se, at sundhedspersonalet ikke har adgang til alarmKlassen eller patientarkiv klassen, som planlagt, samt alle dets planlagte metoder blev kørt. Dette kan ses på figur: 60.



Element	Class, %	Method, %	Line, %
IMG	100% (0/0)	100% (0/0)	100% (0/0)
Beregner	100% (1/1)	100% (14/14)	92% (110/119)
ControllerAlarm	0% (0/1)	0% (0/5)	0% (0/39)
ControllerArkiv	100% (1/1)	100% (8/8)	98% (58/59)
ControllerLogin	100% (1/1)	100% (4/4)	89% (35/39)
ControllerPatientArkiv	0% (0/1)	0% (0/6)	0% (0/18)
ControllerProgramChooser	100% (1/1)	75% (3/4)	80% (8/10)
ControllerProgramChooserNoAlarm	100% (1/1)	100% (3/3)	100% (7/7)
EKGcontroller	100% (1/1)	100% (5/5)	94% (32/34)
FileHandler	100% (1/1)	100% (3/3)	100% (14/14)
Main	100% (1/1)	100% (3/3)	100% (13/13)
Puls_SpO2_TempController	100% (1/1)	100% (5/5)	100% (11/11)

Figure 60: Basis path test Sundhedspersonale

7.2.4 Patient

Under kørsel af Basis path testing af patient fandt vi et problem, at hvis man slettede alt på username pladsen, så ville programmet logge ind. Dette er fordi, at programmet nu skulle lede efter en directory og tjekke om den eksistere. Dette gør den selvfølgelig, da vi har lavet den manuelt. Denne fejl løste vi, ved at tilføje den udkommenterede kontrolstruktur der kan ses på figur: ??.

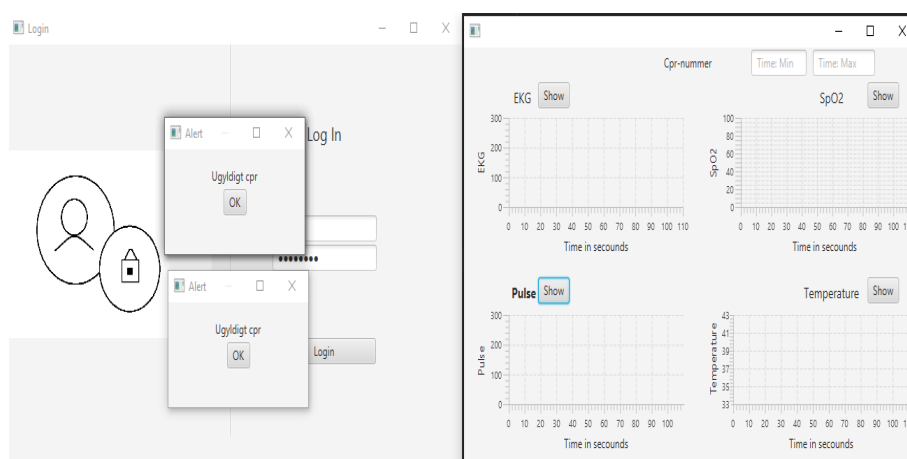


Figure 61: Basis path test patient fejl

Derpå kunne vi gennemføre testen, og man kan se, at patienten kun bruger de metoder og klasser, som var meningen. Dette kan ses på figur: 62

45% classes, 40% lines covered in package 'sample'

Element	Class, %	Method, %	Line, %
IMG	100% (0/0)	100% (0/0)	100% (0/0)
Beregner	100% (1/1)	14% (2/14)	28% (34/119)
ControllerAlarm	0% (0/1)	0% (0/5)	0% (0/39)
ControllerArkiv	100% (1/1)	25% (2/8)	74% (44/59)
ControllerLogin	100% (1/1)	100% (4/4)	89% (35/39)
ControllerPatientArkiv	100% (1/1)	100% (6/6)	90% (18/20)
ControllerProgramChooser	0% (0/1)	0% (0/4)	0% (0/8)
ControllerProgramChooserNoAlarm	0% (0/1)	0% (0/3)	0% (0/6)
EKGcontroller	0% (0/1)	0% (0/5)	0% (0/31)
FileHandler	0% (0/1)	0% (0/3)	0% (0/14)
Main	100% (1/1)	100% (3/3)	100% (13/13)
Puls_SpO2_TempController	0% (0/1)	0% (0/6)	0% (0/12)

Figure 62: Basis path test patient

7.3 Path testing

Målet her er umiddelbart, at prøve alle scenarier af og derved teste om alle klasser, metoder og liner kode bliver brugt. Hvis ikke de bliver brugt, skal man finde ud af hvorfor. Her bemærkes at der mangler at blive kørt fem procent af koden. Det er primært exceptions, som vi havde svært ved at afprøve, men der er også 2 enkelte metoder, der burde været blevet omskrevet. Dette snakker vi om i Implementationskaptitlet under beregner. Dette kan ses på figur: 63.

100% classes, 95% lines covered in package 'sample'

Element	Class, %	Method, %	Line, %
IMG	100% (0/0)	100% (0/0)	100% (0/0)
Beregner	100% (1/1)	100% (14/14)	93% (111/119)
ControllerAlarm	100% (1/1)	100% (5/5)	95% (39/41)
ControllerArkiv	100% (1/1)	100% (8/8)	98% (58/59)
ControllerLogin	100% (1/1)	100% (4/4)	100% (39/39)
ControllerPatientArkiv	100% (1/1)	100% (6/6)	90% (18/20)
ControllerProgramChooser	100% (1/1)	100% (4/4)	100% (10/10)
ControllerProgramChooserNoAlarm	100% (1/1)	100% (3/3)	100% (7/7)
EKGcontroller	100% (1/1)	100% (5/5)	94% (32/34)
FileHandler	100% (1/1)	100% (3/3)	100% (14/14)
Main	100% (1/1)	100% (3/3)	100% (13/13)
Puls_SpO2_TempController	100% (1/1)	100% (5/5)	100% (11/11)

Figure 63: Path test

7.4 Black box

Under black box testing vil vi udfører en state transition testing, hvor vi undersøger om programmet skifter mellem scenerne som planlagt. Vi vil kun undersøge outputtet af koden, da vi laver black box testing. Vi har valgt at dokumentere dette, ved screendumps efter tryk på knapperne, for at skifte scene. Derved kan man teste, om de fører til de pladser de skulle have. Det giver os en kombination af usability testing, opgave baseret, hvor vi kan se om brugerne kan anvende programmet, og om det output vi får, er repræsentativt for reelle målinger. Dvs. vi forsøger gennem funktionel afprøvning, heraf blackbox, at afgøre om følgende punkter fra kravene er opfyldt:

1. Udskrive data fra sensorer
2. Give besked hvis målinger overskrider grænseværdier
3. Præsentere data i en brugergrænseflade
4. Løbende overføre disse data og opdatere dem
5. Finde tidligere data frem.

7.4.1 Login

Læge Når en læge logger ind, skal de blive ført til en scene, hvor de har 4 knapper at vælge imellem. Puls-SpO2Temp, EKG, Alarm og arkiv. Bruger login før udgivelse af program, er



Figure 64: State trasistion testing Læge login

"DR" for læge og "password" for Password. State transistion testing af læge ses på figur: 64 og ??.

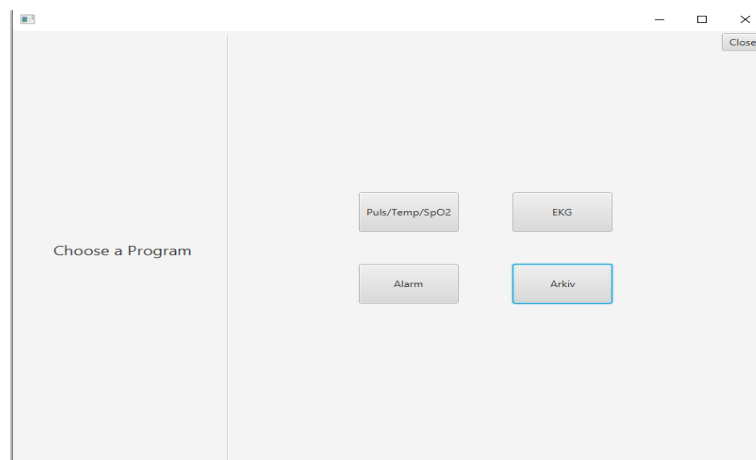


Figure 65: State trasistion testing Læge login positiv

Sundhedspersonale Sundhedspersonale skal have samme scene som lægen. undtagen, at der skal mangle en alarmknap.

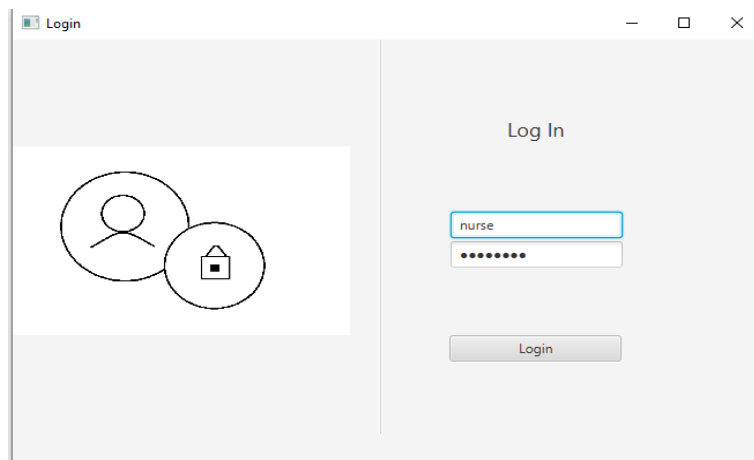


Figure 66: State trasistion testing Sundhedspersonale login

Bruger login før udgivelse af program, er "nurse" for sundhedspersonale og "password" for Password. State transistion testing af sundhedspersonale ses på figur: 66 og 67 .

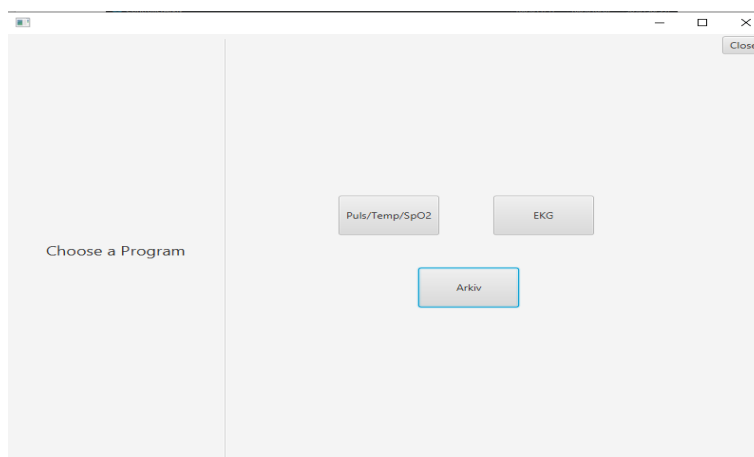


Figure 67: State trasistion testing Sundhedspersonale login positiv

Patient Patienten skal blive ført til sin egne data - dvs. der ikke må vises fra andre. Bruger

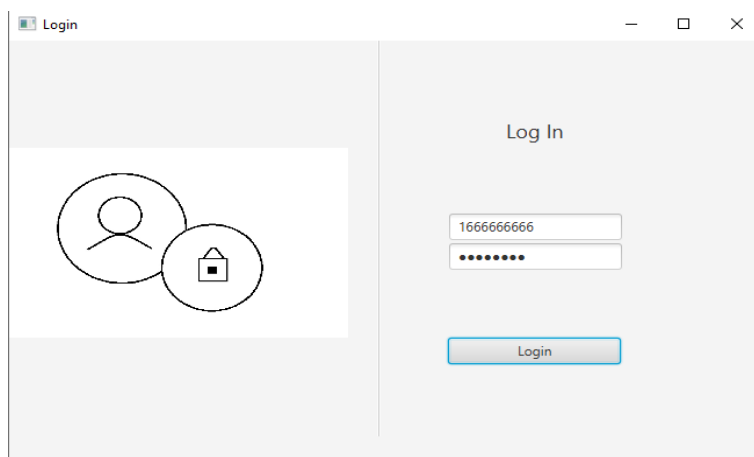


Figure 68: State trasistion testing PatientLogin

login før udgivelse af program, er det indskrevne CPR nr. man har lavet undersøgelserne på og "password" for Password. State transistion testing af Patient ses på figur: 68 og ??.

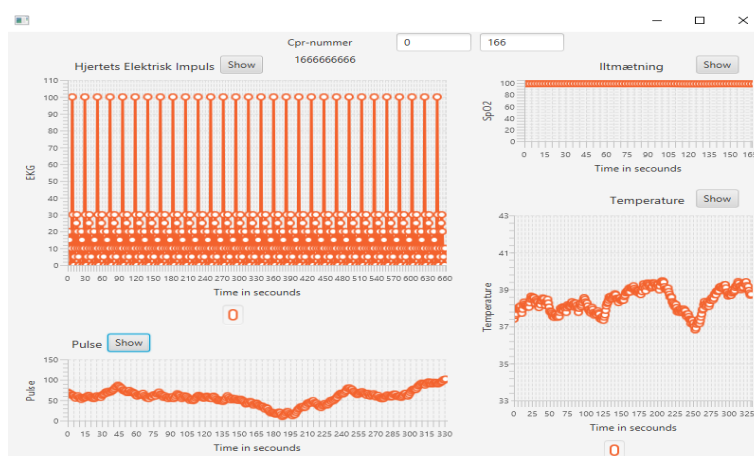


Figure 69: State trasistion testing PatientLogin positiv

Uønsket bruger Skal blive nægtet adgang ses på figur: 68.

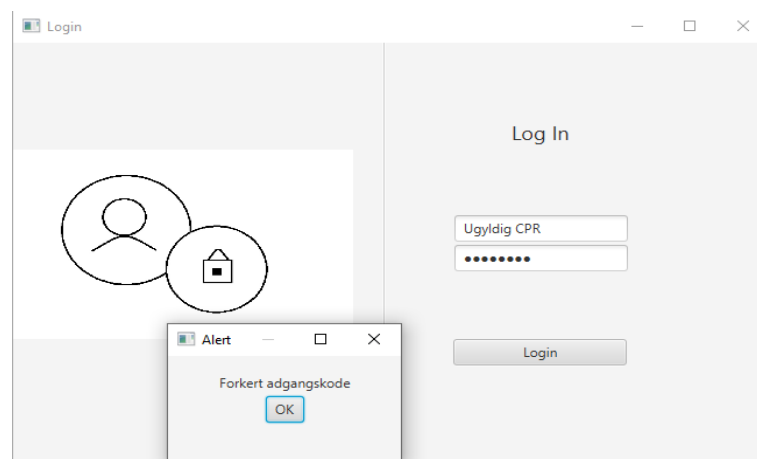


Figure 70: State transition testing Ugyldig login

7.4.2 ChooseProgramScene

Knapperne i scenen hvor man vælger program, skal føre en til dets scener. Her kan vi konkludere at knapperne virkede.

7.4.3 Close funktioner

Close knapperne fungerer alle som de skal, de lukker en scene ned, så man bliver returneret til den forrige.

8 Diskussion

8.1 Sammenligning afprøvningen med personaerne:

8.1.1 Lægen

Resultatet fra afprøvningen med lægen var primært positivt. Det så ud til, at programmet var brugervenligt og har en høj learnability, idet at afprøveren allerede efter første forsøg vidste hvorledes programmet skulle betjenes. Dette er godt, fordi at som beskrevet i personaerne, så er der flere handlinger lægen vil med programmet ift. sygeplejersken og patienten. Så det at lægen synes det var overskueligt, og at learnability var høj, er godt. Dog havde vi underestimeret hvor meget lægen egentlig ville med programmet, idet vi fik nogle konkrete anbefalinger til noget som bør inkluderes. Der blev nemlig ønsket en knap, som tager et screenshot af graferne i arkivet. Udover det, så fokuserede vi primært på at systemet blev simpelt og let for patienten når målingerne fremvises, men dette er også noget vi bør fokusere på ift. foretagelse af målinger. Idet at nogle tekst-labels ikke var iøjnefaldende nok. Dette kan blive et problem, da lægens formål med systemet er at kunne bruge de mange egenskaber der er tilgængeligt for lægen, så derfor skal vi løse problemet. Så vi gør tekst-labelsne tydeligere så lægens og for den sags skyld også sygeplejerskens, interesser med software bedre kan realiseres.

8.1.2 Patient

Resultatet fra afprøvningen med patienten passer godt med det vi opstillede i personaen. I personaen lagde vi nemlig tryk på, at patienten ikke har noget sundhedsvidenskabelig viden. Naturligvis vil der være nogle patienter som besidder en vis viden, men for at gøre det så brugervenligt som muligt, så antager vi at patienten ikke ved noget om det. Derfor har vi simplificeret programmet ned til det balancerede niveau, hvor det er meget forståeligt, men samtidig også indeholder alt den nødvendige data. Og idet at patienten i afprøvningen ikke har kritiseret eller klaget om at det er for kompliceret, så har vi ift. det udført det rigtigt. Dog ikke helt perfekt, da tidsintervalfunktionen ikke er så selvbeskrivende som vi forventede. På baggrund af dette, må vi korrigere den del af softwaren, så det er endnu mere brugervenligt og matcher personaen som jo ingen sundhedsvidenskabelig viden har.

I dette afsnit vil vi diskutere vores endelige løsning. Her vil der ses, hvordan de eventuelle fejlkilder har indflydelse på vores løsning, samt hvilke forbedringer der kan foretages.

8.1.3 Gennemgang af persona-afprøvning

Afprøvning af GUI på personaer er altid en svær sag, da man skal prøve at minimere sin indflydelse på forsøgspersonens valg. Vores situation er ikke blevet nemmere af, at vi har været under corona og testpersonerne derved ikke selv har kunnet styre musen. Dog har vi som sagt prøvet at minimere dette bias, men det kan aldrig helt forsvinde. Under valget af hvem vi ville teste brugergrænsefladen på, valgte vi en læge og en patient, da vi gerne vil se to vidt forskellige tilgange til opgaven. Vi valgte en kvindelig læge i 50'erne, da vi gerne ville teste brugergrænsefladen på en forsøgsperson, der ikke var helt så god til teknik. Under valget af patient, valgte vi en i 20'erne, og dette kan diskuteres om var den rigtige alder at teste på pga. diverse tekniske færdigheder.

8.2 Fejlkilder

I forbindelse med vores afprøvning, stødte vi på en række fejl. Selvfølgelig var der et par mindre fejl, som vi blot rettede, men udover disse var der også andre som vi nu vil nævne. Når målingerne overskred alarm-grænser, gav det en alarm pop-up meddelelse, som var intentionen, men ofte når man lukkede meddelelsen poppede en ny op med det samme. Dette ville være problematisk i henhold til reelle situationer, hvor alarmen er relevant hvis en patient er i fare, så ville det være vigtigt, at kunne tilgå relevante helbredsdata for at kunne behandle patienten optimalt.

Vi fandt også ud af, at vi kunne forbedre vores GUI. Det er også vores intention ved at øge læsbarheden og forståeligheden. Først og fremmest kunne knapperne tilhørende til grafen gøres større og derved mere identificerbare. Sekundært indså vi navngivningen af disse grafer kunne ændres, med formålet at forbedre forståeligheden for patienten, altså hvis SpO2 erstattes med iltmætning og således.

Ved vores egen afprøvning af programmet, stødte vi på et mindre usability problem, og dette problem bestod af at når man vil populære charts i arkivet, reguleres den indtastede max værdi ned til det reelle max af den givne graf. Dette er et problem, da alle graferne ikke har det samme antal målinger dvs. forskellige max værdier. Vi er bekymrede for, at dette kunne resultere i forvirring for brugerne, og at det eventuelt reducerer usability eller blot resulterer i mindre irritationer for ældre brugere.

Under vores udarbejdning af programmet, og på trods af at vi forsøgte at dele hvad vi havde lavet med hinanden, endte vi med at skrive metoder som mindede om hinanden flere steder i programmet. Vi har løbende forsøgt at løse dette problem ved, at konsolidere genbrugelige metoder i beregner klassen og nedarbejde den i næsten alle klasser. Dog under udarbejdningen af implementation afsnittet, er vi stødt på flere muligheder for optimering og simplificering.

8.3 Kode optimering

Efter vejledning fra underviser og i løbet af projektet er vi kommet frem til, at programmet ikke altid overholder DRY-reglen, og der er steder hvor metoder går igen. Nogle af disse steder kan det løses med nedrivning fra fælles klasser, fx. i vores klasser og metoder til at generere tilfældige værdier - og til at opbevare dem.

8.4 Videreudvikling af projektet:

8 timer før deadline for projektet havde deltagerne vejledning med underviser, hvor vi har fået udarbejdet en prioritering, fra høj til lav, over ting, der skal kobles på i løbet af næste projekt, eller rettes til.

1. Kobles en SQL database på backend
2. Løbende hente data fra dem
3. Rette start-knapper til, så der ikke kan oprettes nye objekter hver gang der trykkes
 - (a) Heraf tænkes at lave en Boolean værdi, som ændres ved tryk på en knap, og kun kører, når denne boolean er sand. Det kan gøres vha. pseudokoden "boolean isRunning=false;" og "isrunning = !isrunning;" .

9 Konklusion

Nu kan vi hermed konkludere, at softwaren vi står med nu lever op til de givne krav fra vores stakeholders samt. vores egne forventninger. Ift. stakeholders, vil vi mene, at vi har vi tilfredsstillet Diamongo, ved at udføre alle de givne funktionelle krav. Ift de resterende stakeholders, har vi vha. grundig afprøvning af den grafiske brugergrænseflade så det fungerer optimalt, i forhold til deres krav. Brugergrænseflade passer fint til opgaven, der er ikke for mange antal klik eller for få, samtidigt med at designet er simpelt og effektivt. Under afprøvning af GUI på personaerne, fik vi konkluderet, at vores program havde en høj usability og opfyldtr alle de krav som personaerne ønskene. Dette indebærer høj usability, hvor vores program er meget enkel, samt forskellig funktionalitet for forskellige brugere. Vi kan også konkludere, at selvom vi mødte en del fejlkilder, har vi enten fikset dem, eller planer om at fikse dem.

10 Literaturliste

References

- [1] Jacob Nordfalk. *Objektorienteret programmering i Java udgave 6. - dækker Java 11*. Forlaget Globe A/S.
- [2] Kendall Scott. *UML Explained*. Addison Wesley.

11 Bilag