Roskilde frie børnehave

Af Jonas, Jonathan, Jakob R & Tobias

HCI Tværfagligt Miniprojekt

Fag: SWD, SWK & ITO

Københavns Erhversakademi

Datamatiker 2. Semester

```
public class MainInterface {

    private static ArrayList<Kid> kidList = new ArrayList();
    private static ArrayList<Kid> waitingList = new ArrayList();
    private static ArrayList<Employee> employeeList = new ArrayList();
    private static Manager manager;

    public static JFrame manGUI = new managerGUI("Manager menu");
    public static JFrame empGUI = new employeeGUI("Employee menu");

    public static void load(){
        try{
            kidList.clear();
            employeeList.clear();
            waitingList.clear();
            Scanner kidInStream = new Scanner(new File(".\\resources\\kid_Info.txt"));
            Scanner empInStream = new Scanner(new File(".\\resources\\employee_Info.txt"));
            Scanner manInStream = new Scanner(new File(".\\resources\\manager_Info.txt"));
            Scanner wordStream;
            String temp;

            while (kidInStream.hasNextLine()) {
                temp = kidInStream.nextLine();
                wordStream = new Scanner(temp);
                //nedenstående fjerner id nummeret fra filen
                wordStream.nextInt();
                kidList.add(new Kid(wordStream.next().replaceAll("_", " "), wordStream.next().replaceAll("_", " "),
            }

            while (empInStream.hasNextLine()) {
                temp = empInStream.nextLine();
                wordStream = new Scanner(temp);
                //nedenstående fjerner id nummeret fra filen
```

Indhold

IT Organisationen (ITO) 3
Feasibility study 3
Software Development 4
Use Cases/descriptions 4
Use Case Diagram 4
Use Case Brief 4
Use Case Casual 4
Use Case Fully Dressed 5
FURPS 6
Unified Modeling Language (UML) 7
Design Class Diagram 7
Sequence Diagram (bilag 1) 7
System Sequence Diagram (bilag 2) 7
Activity Diagram 8
Software Construction 9
E/R Diagram 9
Program prototyping 9
Kodeeksempel 1: 9
Kodeeksempel 2: 10
Kodeeksempel 3: 10
Link til koden 11
Bilag 13
Bilag 1 (Sequence Diagram) 13
Bilag 2 (System Sequence Diagram) 13

Link til koden - https://github.com/jacqques/Roskilde-boernehave.git

IT Organisationen (ITO)

Feasibility study

Det er et relativt simpelt produkt vi har med at gøre, det er en database. Dette betyder, at vi i et teknisk øjemed ikke burde støde ind i mange problemer. Da vi allerede har arbejdet med java før og lavet programmer der minder meget om nuværende projekt, er det heller ikke et problem at producere produktet. Produktet kommer også til at være meget sikkert, da alt data er offline og det er nemt at lave en backup.

Med hensyn til, om projektet er økonomisk feasible, så er produktet simpelt og produktet kan derfor hurtigt udvikles. I forhold til omkostningerne ved udvikling af produktet, er produktet billigt at producere, billigt at vedligeholde og billigt at opretholde. Det er tilfældet da programmet ikke indeholder større kompleks logik. Hvis det i fremtiden bliver besluttet, at man gerne vil have nye funktioner, kan det øge omkostningerne ved vedligeholdelse og udvikling. Da dette projekt er et administrativt produkt, findes der sandsynligvis et tilsvarende produkt på markedet. Der skal derfor undersøges, om der allerede er lignende programmer på markedet.

De operationelle krav, der er til produktet, er ikke særligt komplicerede og derfor burde der ikke være nogle problemer. Ledelsen er fuldt ombord på produktet og vi kan nemt gå i dialog angående mulige problemstillinger der kunne opstå. Dette gælder også efter produktet er blevet deployed, hvis der skulle findes fejl. vi forventer ikke, at der kommer til at være problemer efter launch og vi forventer, at det kommer til at løse det administrative behov. Projektet kommer også til at være en fordel for organisationen, da det hele bliver digitalt i stedet for i papirform.

En åben problemstilling for produktet er persondataloven. Dette er forsøgt løst, da programmet kører offline på én computer, hvor alle der har adgang til de følsomme data, arbejder for børnehaven og har et personligt login. Der skal også tages forbehold til licenser og lignende, i forhold til hvem der ejer programmet. Vi sælger programmet og licensen til børnehaven og de får derefter ansvaret for vedligeholdelse og andre udgifter der kunne fremkomme. Efter en tilfredsstillende overdragelse, produktet lever op til krav og operer uden problemer, vil vi kun vider udvikle/fjerne bugs mod betaling. Dette er tilfældet, da vi som studerende ikke kan blive ved med at arbejde i en uendelighed gratis.

Som tidligere nævnt foreslås det at undersøge om der allerede eksisterer programmer, der kan løse de administrative problemer kunden har. Fordelene ved at de får lavet deres eget program er, at det kommer til at være mere simpelt og derved mere brugervenligt og de kommer til at eje rettighederne, så der er mindre chance for pludselig at miste adgangen i fremtiden.
Det kan kun anbefales at udvikle programmet, hvis der ikke i forvejen findes udviklede software, der giver en økonomisk fordel og løser den administrative problemstilling.

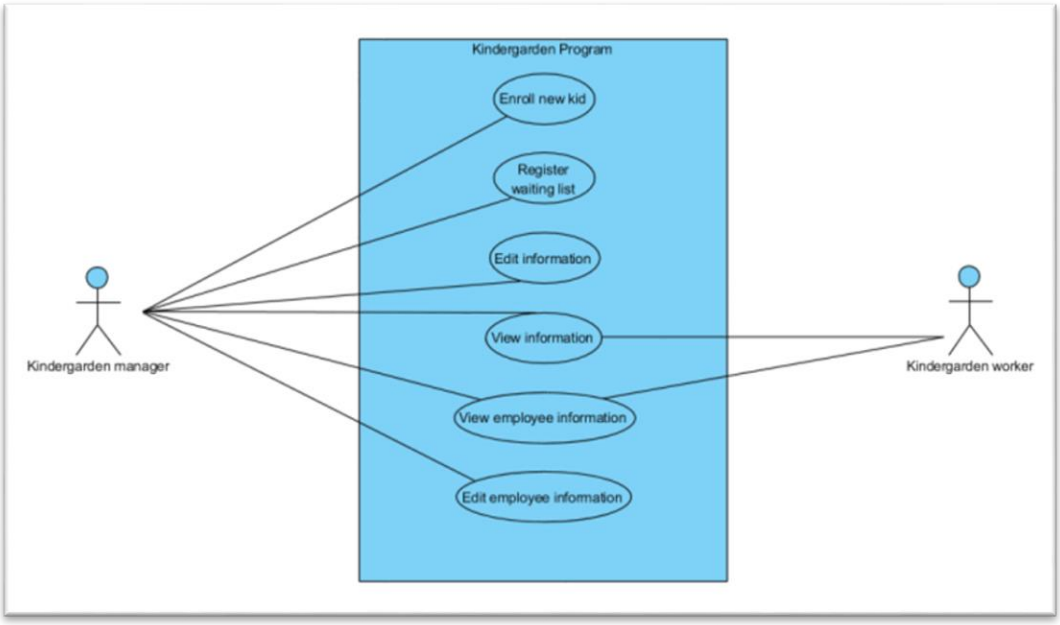
For at konkludere om projektet er et go eller no go, vil det afhænge af nuværende programmer der eksisterer på markedet. Hvis der allerede eksisterer et program der lever op til deres behov, som er økonomisk feasible, vil det anbefales at bruge den software i stedet for at udvikle nyt. Det kommer til at være nemmere og hurtigere i forhold til hvor hurtigt programmet kan tages i brug og en bedre økonomisk beslutning. Der vil også være større mulighed for flere funktioner og vedligeholdelse på lang sigt. Hvis der ikke findes en allerede udviklet løsning på markedet, vil det være muligt at udvikle et nyt.

Software Development

Use Cases/descriptions

Use Case Diagram

Kigger man på det ovenstående use case diagram, kan man hurtigt se at programmet er centreret omkring manageren Sandra, samt nogle funktioner til medarbejdere i børnehaven. De to funktioner som medarbejderne kommer til at arbejde med, er meget simple og kan også nemt bruges af manageren. Det eneste der har været lidt snak om i.f.t. dette diagram er om at når man skal registrere sig til ventelisten og når der skal oprettes et nyt barn i programmet, skal det være den samme funktion? Det er på nuværende tidspunkt to forskellige metoder.



Use Case Brief

Vi har valgt at lave vores brief use case, over vores case ”View information”:

Brugeren åbner vores program og logger ind, herefter vælger de *Information,* så vælger de *Se information om børn*, vælger hvilket barn de vil se information på ud fra barnets identifier. Brugeren ser informationen og afslutter programmet.

Use Case Casual

Vi har valgt at lave vores casual, over vores case ”Register waiting list”:

Main success scenario:

- Lederen åbner programmet og logger ind, vælger *registrer til venteliste*. Lederen bliver promptet for information og udfylder gældende data: når alt data er indskrevet, promptes der igen for om den indskrevne data er korrekt. Dataene er korrekte så lederen gemmer data og afslutter.

Alternative scenarios:

a) Indtastet data er forkert.
    a. 1) Ændrer dataene
    a. 2) Accepterer dataene er korrekte og gemmer.
b) Efter date er gemt findes der fejl.
    b. 1) Lederen vælger *Rediger venteliste*.
    b. 2) Ændrer dataene til den korrekte information.
    b. 3) Gemmer den nye data og afslutter programmet.

Use Case Fully Dressed

Vi har valgt at lave vores fully dressed, over vores use case ”Register waiting list”:

Use case: Register waiting list.

Scope: Børnehave program.

Level: User goal.

Primary actor: Lederen.

Stakeholders and interest:

- Lederen: Ønsker let brug af programmet, ved simpel indtastning af data.
- Pædagoger: Ønsker at børnene nemt kan oprettes i systemet, så der ikke er fremtidige problemer med dem når de afleveres eller lign.
- Forældre: Ønsker nem og tryg proces ved indskrivning af deres barn/børn.

Preconditions:

- Lederen har fået en anmodning om indskrivning af et barn til.

Success guarantee (postcondition):

- Barnet er blevet indskrevet på ventelisten.

Main success scenario (basic flow):

1) Lederen modtager indskrivnings anmodning og skal derefter indskrive et barn på ventelisten.
2) Lederen starter programmet og vælger *Registrer til venteliste*.
3) Lederen indtaster barnets information.
4) Lederen gemmer barnets information og afslutter programmet.

Extension (alternative flow):

a) Lederen skriver forkert information.

a. 1) Lederen vælger *Rediger venteliste information*.
a. 2) Lederen vælger hvilket barn der skal redigeres, via barnets identifier.
a. 3) Lederen retter sine data fejl og gemmer.

b) Der er ingen børn på ventelisten.
b. 1) Barnet bliver indskrevet i børnehaven.
c. 2) Lederen får besked om at barnet er oprettet og informerer forældrene.

Special requirements:

- Lederen skal have en PC.
- Lederen skal have en form for database over de nuværende børn på ventelisten.

Frequency of occurrence:

- Lederen ville skulle registrere nye børn på ventelisten, hver gang der kommer en ny anmodning om at et barn vil starte i børnehaven.

Open issues:

- GDPR

FURPS

Functionality:

- Vores program skal kunne oprette og redigere venteliste med børn for at skabe et overblik over børnehavens nuværende medlemmer og ansatte. Programmet kan holde styr på hvert enkelt barn, samt forældrenes information. Man skal kunne se hvilke børn der går i børnehaven og hvilke børn der er på venteliste. Manageren og de ansatte skal kunne trække lister over børnene i børnehaven og ventelisten.

Usability:

- Vores program skal være let at bruge og man skal ikke igennem mange trin. Programmet skal være let for de ansatte at bruge og de skal let kunne vælge mellem de forskellige menuer/funktioner

Reliability:

- Man skal i vores program kunne gemme til en liste over børn i børnehaven i en fil, samt kunne skrive til en fil ved nye oprettelser af børn.

Performance:

- Programmet skal køre med en minimumforsinkelse på 500ms.

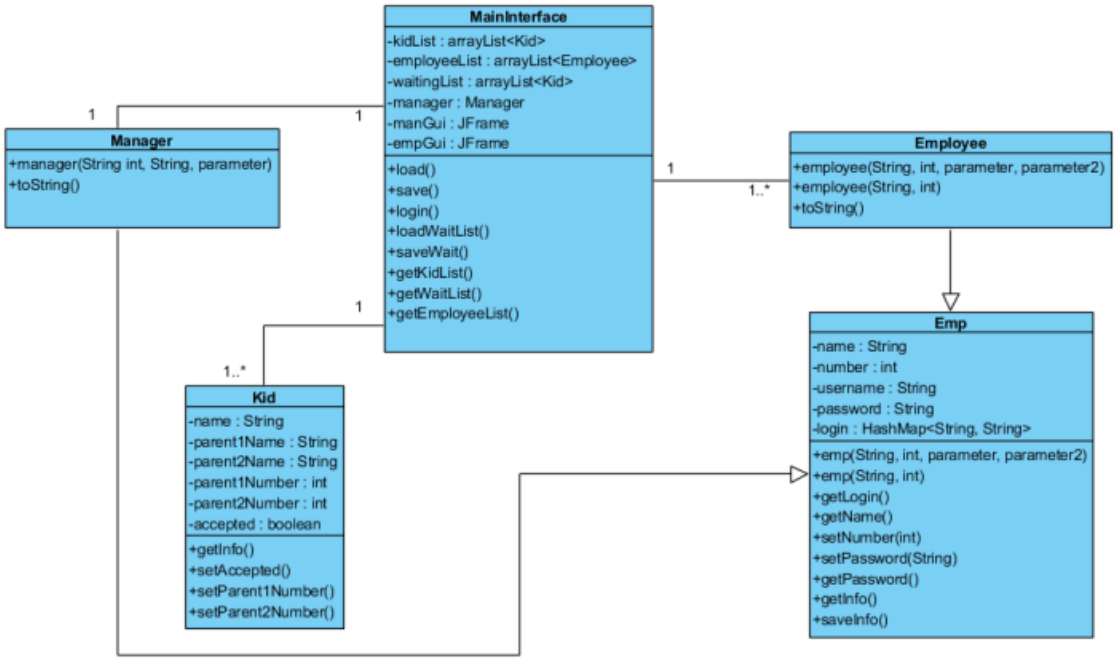
Supportability:

- Vi skal senere kunne implementere filhåndtering, uden større ændringer i koden.

Unified Modeling Language (UML)

Design Class Diagram

Vi har valgt at lave et design klasse diagram, fordi det giver en god illustration over hvilke klasser, attributter og associationer som vi har tænkt os at bruge i vores programmering. Vi har valgt, at vi vil lave 5 klasser, hvor vi i hver klasse har beskrevet hvilke datatyper og metoder som vi vil bruge. De enkelte klasser har så associationer til hinanden, afhængigt af om de integrerer med hinanden 1 til 1 eller 1 til mange.



Sequence Diagram (bilag 1)

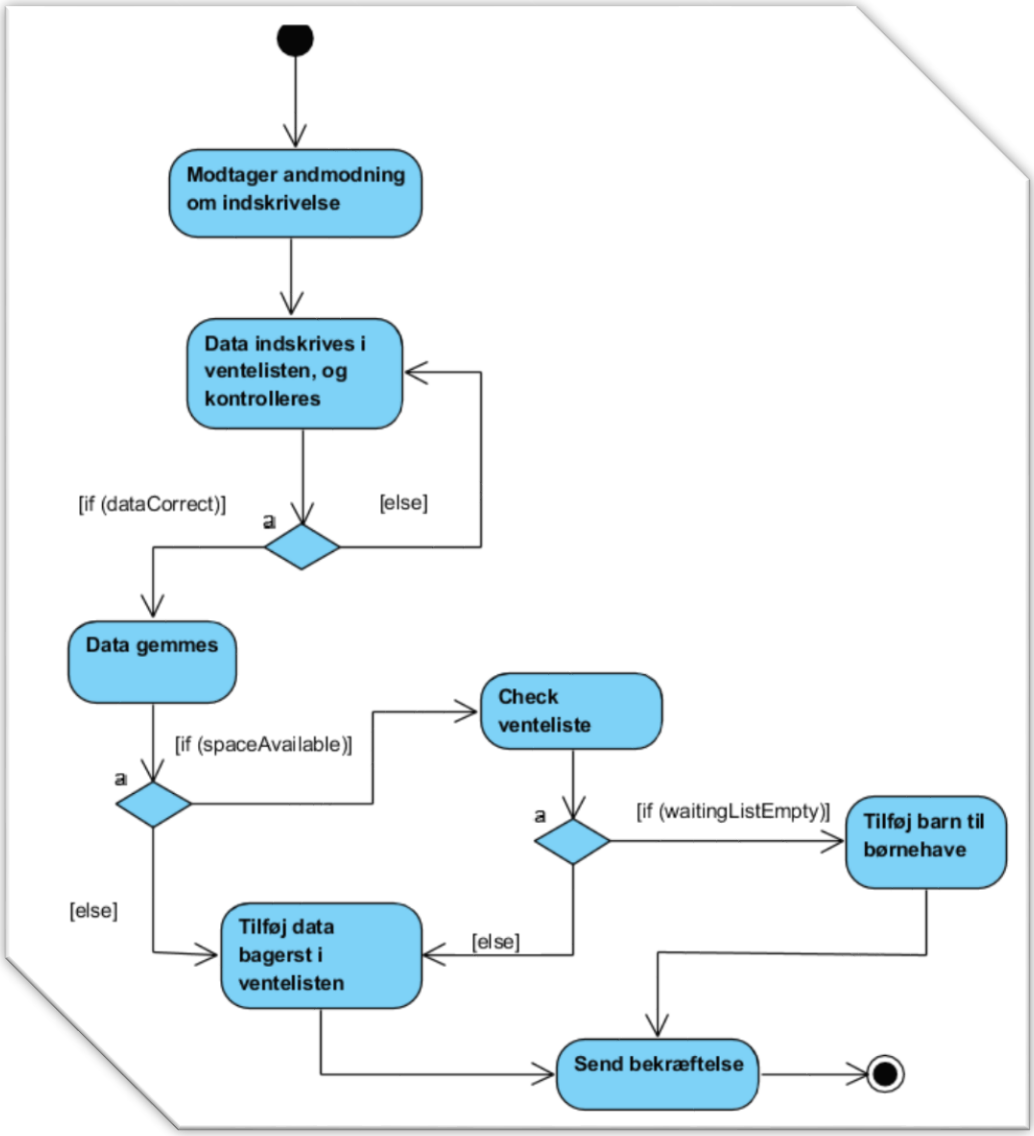
Vores sekvens diagram (se bilag 1) har vi lavet over vores *load()* metode i vores *MainInterface* klasse. Vi har valgt at lave sekvensdiagrammet ud fra vores GUI interface, men vi befinder os stadigvæk i MainInterface klassen i Java. Vores *load()* metode fra rfbGUI klassen, sørger for at hente data omkring de nuværende børn i børnehaven fra vores filer. Sekvensdiagrammet giver et godt overblik over vores objektinteraktioner, som vi vil bruge til at få vores load metode til at virke. Diagrammet viser de datastrømme, som er med til at få vores *load()* metode til at fungere.

System Sequence Diagram (bilag 2)

Vores system sequence diagram (se bilag 2) visualiserer vores use case *Register waiting list*. Vores manager vælger at tilføje til børnehavens venteliste *AddWaitingList()*, hvor manageren derefter bliver promptet til at indtaste data om barnet *EnterKidData*, hvorefter manageren udfylder informationen. Manageren bliver herefter promptet, om det indtastede data er korrekt og manageren bekræfter dataene er korrekte.

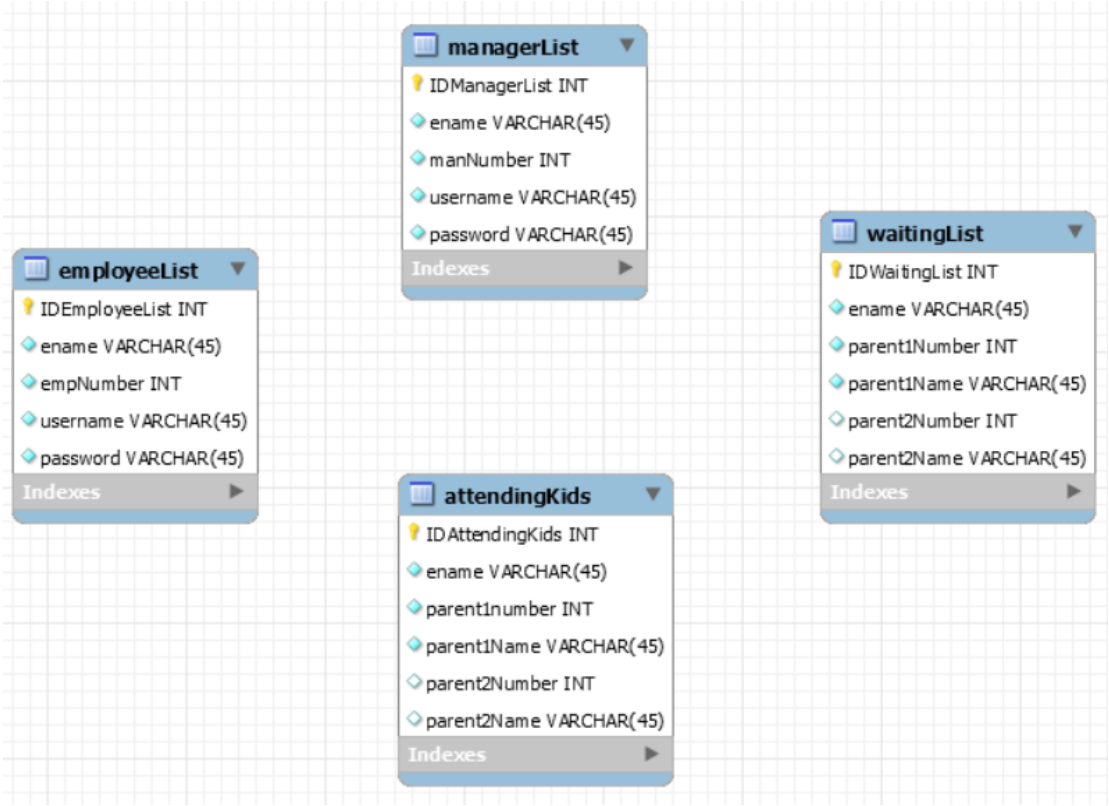
Activity Diagram

Vores aktivitetsdiagram har vi lavet over vores use case *Enroll new kid*. Aktivitetsdiagrammet starter, hvis manageren modtager information om, at der er et nyt barn, som skal indskrives i børnehaven. Derefter tjekkes der, om dataene er korrekte og derefter eksekveres metoden, som vælger om barn skal indskrives i børnehaven eller skal på venteliste.



Software Construction

E/R Diagram



Program prototyping

Kodeeksempel 1:

Dette kodeeksempel er taget fra klassen emp som er en superklasse for manager og employee.

Når vi overrider konstruktøren gælder det både for vores manager og employee.

På linje 15 og linje 23 bruger vi et *HashMap* til at gemme login information.

```
10      public Emp(String name, int number) {
11          this.name = name;
12          this.number = number;
13          this.username = "null";
14          this.password = "null";
15          login.put(this.username, this.password);
16      }
17
18      public Emp(String name, int number, String username, String password) {
19          this.name = name;
20          this.number = number;
21          this.username = username;
22          this.password = password;
23          login.put(this.username, this.password);
24      }
```

Kodeeksempel 2:

På dette kodeeksempel viser vi vores login metode, hvor brugeren skal indtaste username/password.

I vores for loop tjekker vi, om vores login information passer til vores manager eller employee. Vi loop'er igennem vores HashMap for at tjekke om det data passer med en af brugerne til programmet (manageren/employee). Hvis dataene passer, så sætter man den gældende GUI til at være visible, på linje 69 og 78 gøres det så det nye vindue åbner i max størrelse (*MAXIMIZED_BOTH*).

I vores *if* statement som starter på linje 82, tjekker vi om brugeren har fået adgang – hvis brugeren ikke har adgang kommer der et popup vindue med vores message.

```
66      for (Employee employee : employeeList){
67          if (employee.getLogin().containsKey(tempUsername)) {
68              if (employee.getLogin().containsValue(tempPassword)) {
69                  empGUI.setExtendedState(JFrame.MAXIMIZED_BOTH);
70                  empGUI.setVisible(true);
71                  access = true;
72                  break;
73              }
74          }
75      }
76      if (manager.getLogin().containsKey(tempUsername)) {
77          if (manager.getLogin().containsValue(tempPassword)) {
78              manGUI.setExtendedState(JFrame.MAXIMIZED_BOTH);
79              manGUI.setVisible(true);
80              access = true;
81          }
82      }if (!access){
83          JOptionPane.showMessageDialog( parentComponent: null,  message: "Username or password incorrect, try again.");
84      }
```

Kodeeksempel 3:

På linje 33 har vi gjort sådan, at det kun er det vindue som brugeren står i, som lukker (ved tryk på X, Alt+F4)

I vores rfbGUI.java på linje 20 står der *EXIT_ON_CLOSE*, dette betyder at når dette vindue lukker, så lukker hele programmet.

På linje 41 bruger vi metoden som hedder *addActionListener*, som holder øje med om knappen bliver trykket på og eksekvere koden i *try/catch* blokken

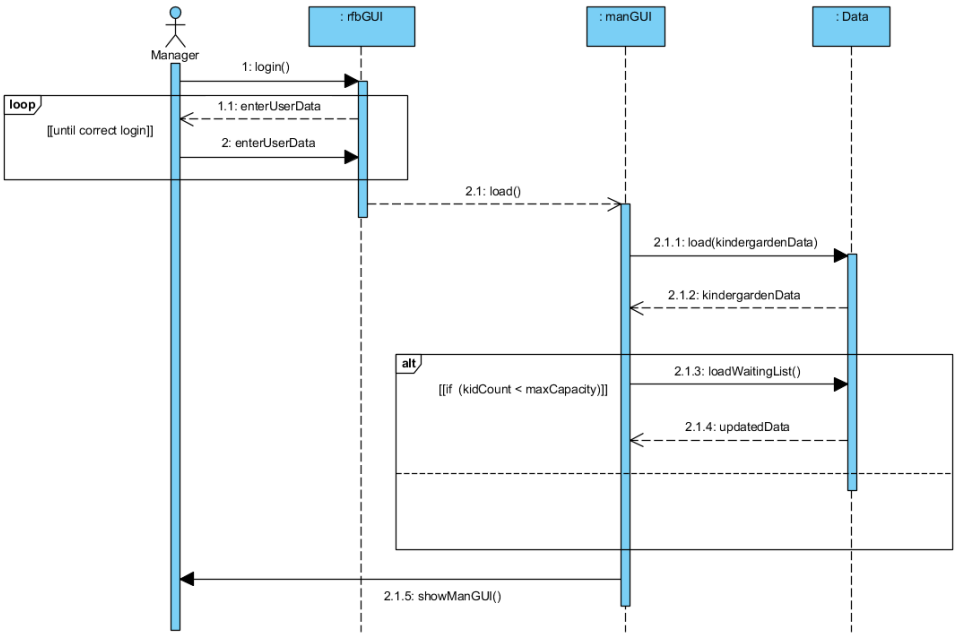
```
33          this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
34          this.setContentPane(managerPanel);
35          this.pack();
36
37          methods.load();
38          methods.loadWaitingList();
39          methods.save();
40
41          viewInfoButton.addActionListener(new ActionListener() {
42              @Override
43              public void actionPerformed(ActionEvent e) {
44                  try {
45                      displayInfoGUI.setExtendedState(JFrame.MAXIMIZED_BOTH);
46                      displayInfoGUI.setVisible(true);
47                  } catch (Exception ex) {
48                      JOptionPane.showMessageDialog( parentComponent: null,  message: "Something went wrong.");
49                  }
50              }
51          });
```

Link til koden - https://github.com/jacqques/Roskilde-boernehave.git

Bilag

Bilag 1 (Sequence Diagram)

Tilbage til side 7



Bilag 2 (System Sequence Diagram)

Tilbage til side 7