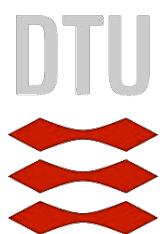


# **QuickConnect - Livechat**

**af gruppe 17**



DANMARKS  
TEKNISKE  
UNIVERSITET



**CDIO - Final**

17. juni 2016

# **Titelblad**

Titel: QuickConnect - Livechat

**Projekt type:** CDIO - Final

Gruppe nr.: 17

Uddannelsesinstitution: Danmarks Tekniske Universitet – DTU

Fag og retning: Videregående programmering - Softwareteknologi

Dato for aflevering: 17.06.2016

Denne rapport er afleveret via Campusnet (der skrives ikke under)

Denne rapport indeholder 65 sider.

## Udarbejdet af:

	Harun	Ibrahim al-Bacha	Samil Batir	Tolga Cetin	Umais Usman Shoaib
Ahmad Almajedi	Abd Ullah				

Studie nr.: s153317 s150366 s118016 s153191 s133959 s153403

Vejledere: Stig Finn  
Høgh Gustafsson



# Indholdsfortegnelse

Timeregnskab .....	5
1. Indledning .....	6
2. Problemfelt.....	7
2.1. Problemanalyse .....	7
2.2. Problemformulering .....	7
2.3. Rapportens opbygning .....	8
3. Analyse.....	10
3.1. Kravspecifikation .....	10
3.2. Use case beskrivelser.....	11
3.3. Use case diagram.....	20
3.4. BCE model.....	22
3.5. Klasse diagram.....	25
4. Implementering .....	27
4.1. Forbindelse til databasen.....	27
4.2. Mail.....	32
4.3. Facebook.....	34
4.4. Vigtige ting i forhold til koden.....	36
4.5. Threads .....	37
5. Design-dokumentation.....	39
5.1. ER – diagram .....	39
5.2. Schema diagram .....	41
5.3. Design-sekvens diagram.....	43
5.4. System-sekvens diagram .....	48
5.4.3. Login .....	51
5.4.4. Ændring af nickname .....	53
5.4.5. Venneanmodning.....	54
5.4.6. Deaktivér bruger.....	55
5.5. 3-lagsmodellen .....	57
5.6. Nem-ID validering .....	58
6. Test.....	59
6.1. Test case 1: Ændre password test.....	59
7. Konklusion .....	63

8. Konfiguration.....	64
9. Referencer .....	65
9.1. Hjemmesider.....	65
9.2. Programmer.....	65

# Timeregnskab

Følgende er en oversigt over et estimat af hvor mange timer hvert gruppemedlem har brugt på projektet samt hvilken del af projektet disse timer er blevet brugt på.

Overblikket over antal timer som hver person har brugt har været vanskelig af primært to årsager.

Først og fremmest fordi at alle medlemmer af gruppen næsten udelukkende har siddet sammen og arbejdet på alle dele af projektet.

Dernæst fordi der ikke har fundet en reel notering sted for hvert medlem over de antal timer vedkommende har brugt. Dette har dog heller ikke været det mest væsentlige, da naturen af arbejdet ikke krævede det. I en reel arbejdssituation med lønninger mv. så ville dette dog være påkrævede, således at hver person som arbejder på projektet skal notere de antal timer han/hun bruger på arbejdet og ligeledes præcist hvad personen har arbejdet på.

Vi mener dog stadig at nedenstående tabel afspejler fordelen af hvert gruppemedlems bidrag samt antal timer brugt på de forskellige delelementer af projektet.

	Database	Analyse	Kode	Dokumentation	Test	Total(timer)	Procent(%)
Ahmad	4	3	43,5	5	-	55,5	17,08
Harun	3	5	37	6	2	53	16,3
Tolga	2	5	41,5	6	-	54,5	16,77
Samil	2	1	46,5	3	3	55,5	17,08
Ibrahim	3	5	39	7	-	54	16,62
Umais	3	6	36,5	7	-	52,5	16,15
Total(timer)	17	25	244	34	5	325	

# 1. Indledning

I denne rapport vil man kunne læse om et livechat-program, hvor 2 eller flere personer kan kommunikere med hinanden. Dette sker ved at programmere i Java, hvor vi forbinder det med en database. Vi har oprettet nogen forskellige tabeller i databasen, som skal sørge for at lagre forskellige attributter. Gennem vores Java kode, kan vi så sende forespørgsler til databasen. Dette sker eksempelvis, når man skal logge ind i programmet, hvor systemet vil spørge databasen, om brugeren eksisterer.

I programmet skal man blandt andet kunne følgende:

- Oprette bruger (e-mail validering, Facebook validering)
- Logge ind
- Tilføje venner
- Chatte med andre brugere
- Slette kontakter
- Blokere kontakter
- Ændre brugeroplysninger
- Oprette grupper
- Slette grupper
- Smide ud af grupper
- Forlade grupper osv...

For at kunne realisere disse krav, har vi til at starte med analyseret problemstillingen, og derefter designet en løsning på det. Vi har valgt, at analysere problemstillingen, ved blandt andet at lave en BCE model og en domænemodel. Derefter lavede vi vores database og begyndte at kode i Java. Til sidst lavede vi så design delen, ved at designe ved hjælp af eksempelvis Design Sekvens Diagram og System Sekvens Diagram.

## **2. Problemfelt**

### **2.1. Problemanalyse**

Da vi overvejede hvilken slags projekt vi ville lave, syntes vi, at vi skulle fokusere på noget, som kan bruges dagligt og samtidig af mange. Vi har haft flere ideer, såsom biograf reservations-system, pengeautomat og et chatsystem. Vi har valgt at fokusere på et chatsystem, da vi synes at det var mere spændende, samtidig med at vi kunne inddrage flere dele af pensum. Projektet går ud på at lave et chatsystem, hvor to eller flere personer kan kommunikere med hinanden. Et chatsystem består typisk af brugere som kan registrere sig og logge ind, dermed vil brugeren tilføje kontakter, som han/hun kan chatte med. Her skal man overveje hvilke funktioner programmet skal have, og hvordan man vil sikre et sikker system for brugerne.

### **2.2. Problemformulering**

På baggrund af vores problemanalyse, lyder vores problemformulering således:

- Hvordan kan vi sikre et sikkert system for brugerne?
- Hvordan kan vi verificere brugeroplysninger?
- Hvordan kan vi få 2 eller flere personer til at kommunikere med hinanden?

## **2.3. Rapportens opbygning**

Rapportens struktur og opbygning er som følgende:

### Kapitel 1 – Indledning:

*En kort introduktion til projektet og hvad det går ud på, samt hvad man kan forventer at få nærmere information om i rapporten.*

### Kapitel 2 – Problemfelt:

*Problemstillingen belyses og afgrænses og en problemformulering udformes som hele projektet skal sigte imod at løse.*

### Kapitel 3 – Analyse:

*Indeholder alle de relevante informationer, beskrivelser og modeller for at forstå de krav som vores design af programmet må leve op til og som senere vil forme implementeringen.*

### Kapitel 4 – Design:

*Designet af vores program og selve systemet, illustreret vha. primært 3 modeller nemlig, domænemodellen, BCE modellen og klasse diagrammet. Dette forbereder og klargør implementeringsfasen.*

### Kapitel 5 – Implementering:

*Indeholder de vigtigste udpluk fra implementeringsdelen, som viser hvordan vi praktisk har iværksat det design som vi tidligere kom frem til igennem vores analysedel.*

### Kapitel 6 – Dokumentation:

*Her dokumenteres og forklares hvordan objekterne i programmet kommunikere vha. sekvens modeller samt strukturen i hele vores program samt andet relevant dokumentation.*

## Kapitel 6 – Test:

*De vigtigste og centrale test cases forklaret i dette afsnit for at få et indblik i en del af den måde vi har testet vores program.*

## Kapitel 6 – Konfiguration:

# 3. Analyse

## 3.1. Kravspecifikation

- Det skal være muligt at køre på alle computer-enheder som understøtter Java
- Slut-brugeren skal kunne registrer sig med et ønsket brugernavn, email og password
- Slut-brugeren skal kunne logge ind efter, og kun efter vedkommende har bekræftet sin email
- Slut-brugeren skal kunne kommunikere med andre brugere, og lave grupper hvor vedkommende kan snakke i gruppevis
- Slut-brugeren skal kunne sende venneanmodninger til andre brugere, således at hvis de accepterer venneanmodningen, vil begge brugere kunne se om de er online (logget på) eller offline (logget af)
- Ligeledes skal slut-brugeren kunne blokere andre brugere således at de ikke kan skrive beskeder eller sende venneanmodninger til vedkommende
- Slut-brugeren skal kunne ændre sit kaldenavn og password når vedkommende ønsker det
- Slut-brugeren skal kunne deaktivere sin bruger når vedkommende ønsker dette
- Der ønskes en anden form for registrering, f.eks. Google eller Facebook
- Der ønskes censur på selvvalgte ord
- Der ønskes notification for nye modtagne beskeders

### 3.2. Use case beskrivelser

<b>Use Case:</b> Registrering
ID: 1
<b>Brief Description:</b> <u>QuickConnect registrering:</u> Brugere ønsker at registrere sig med QuickConnect's registrering <u>Facebook registrering:</u> Brugeren kan ligeledes vælge at registrere sig gennem sin Facebook konto.
<b>Primary actors:</b> For enkel bruger 1. For hele programmet uendelig mange.
<b>Secondary actors:</b>
<b>Preconditions:</b> <ol style="list-style-type: none"><li>1. Forbindelse til netværk.</li><li>2. Programmet opstartes.</li><li>3. Der trykkes på registrer.</li></ol>
<b>Main flow:</b> <ol style="list-style-type: none"><li>1. Alle felter udfyldes.<ol style="list-style-type: none"><li>a. Brugernavn</li><li>b. Password og gentagelse af password</li><li>c. Email</li><li>d. Fødseldagsdato</li><li>e. Vilkår og betingelser accepteres</li></ol></li><li>2. Der trykkes på registrer</li></ol>

3. Bruger indtaster kode som modtages fra email  
Facebook registrering
4. Brugeren registrerer sig vha. Facebook.
5. Brugeren indtaster brugernavn og password
6. Bruger indtaster modtaget kode
  
7. Programmet tjekker om brugernavnet
  - a. Indeholder mellem 4-24 tegn
  - b. Brugernavnet indeholder mindst 1 tal, et stort bogstav og et lille bogstav
8. Tjekker om password er
  - a. Mellem 8-24 tegn
  - b. Begge password er ens
  - c. Passwordet indeholder tal, bogstav og special tegn
9. Programmet tjekker om email korrekt
10. Beregner alder

---

**Post conditions:**

1. Han er nu registreret som brugere af programmet

---

**Alternative flows:**

1. Programmet crasher
2. Databasen går ned
3. Intet internet

---

**Use Case: Log In**

---

ID: 2

<b>Brief Description:</b>
Brugeren ønsker at logge ind, for at bruge programmets funktioner.
<b>Primary actors:</b>
1 Bruger
<b>Secondary actors:</b>
<b>Preconditions:</b>
1. Use case 1 er opfyldt
<b>Main flow:</b>
1. Brugeren indtaster sit brugernavn og password ind. 2. Brugeren klikker på knappen log ind 3. Systemet tjekker i databasen om brugernavn og password kombinationen eksistere for en given id, i databasen
<b>Post conditions:</b>
1. Han er nu logget ind og kan bruge programmets funktioner.
<b>Alternative flows:</b>
1. Programmet crasher 2. Databasen går ned 3. Intet internet

<b>Use Case: Ven Anmodninger</b>
ID: 3
<b>Brief Description:</b> Brugere ønsker at tilføje en ny ven til vennelisten.

**Primary actors:**

1 Bruger

**Secondary actors:**

**Preconditions:**

1. Use case 2 er opfyldt

**Main flow:**

1. Brugeren trykker på + og derefter tilføj venner
2. Indtaster vedkommendes brugernavn i textfeltet
3. Systemet tjekker om:
4. Brugeren eksistere
5. Brugeren er blokeret
6. De ikke allerede er venner
7. Der ikke er blevet sendt en venneanmodning
8. Hvis overstående er opfyldt så sendes en venneanmodning

**Post conditions:**

1. Anmodningen bliver sendt

**Alternative flows:**

1. Systemet crasher
2. Venneanmodning bliver ikke sendt

**Use Case:** Ændring af password

**ID:** 4

**Brief Description:**

Bruger ønsker at skifte sit password.

**Primary actors:**

1 Bruger

**Secondary actors:****Preconditions:**

1. Use Case 2 er opfyldt.

**Main flow:**

1. Der trykkes på min profil og derefter indstillinger
2. Brugeren trykker på ændrer password og indtaster nyt password
3. Systemet tjekker om passwordet indeholder de forventet kriterier
4. Er mellem 8 og 24 tegn
5. Indeholder et stort bogstav eller et lille bogstav
6. Indeholder et tal
7. Systemet tjekker om det gamle password er korrekt indtastet
8. Systemmet tjekker om det nye password er indtastet korrekt 2 gange

**Post conditions:**

1. Passwordet er ændret i databasen

**Alternative flows:**

1. Programmet crasher
2. Databasen går ned
3. Intet internet

**Use Case:** Ændring af nickname

ID: 4

<b>Brief Description:</b>
Bruger ønsker at skifte sit nickname
<b>Primary actors:</b>
1 Bruger
<b>Secondary actors:</b>
<b>Preconditions:</b>
1. Use Case 2 er opfyldt.
<b>Main flow:</b>
1. Forbindelse til netværk. 2. Use Case 2 er opfyldt. 3. Der trykkes på min profil, derefter indstillinger og ændre nickname 4. Systemet tjekker om længden af nickname er 0 eller et mellemrum
<b>Post conditions:</b>
1. Nickname ændres i database
<b>Alternative flows:</b>
1. Programmet crasher 2. Databasen går ned 3. Intet internet

<b>Use Case:</b> Oprette gruppe
<b>ID:</b> 5
<b>Brief Description:</b>
Bruger ønsker at oprette en gruppechat
<b>Primary actors:</b>

1 Bruger
<b>Secondary actors:</b>
<b>Preconditions:</b>
1. Use Case 2 er opfyldt.
<b>Main flow:</b>
<ol style="list-style-type: none"> <li>1. Der trykkes på grupper og på +             <ol style="list-style-type: none"> <li>a. Der trykkes på opret gruppe</li> <li>b. Der indtastes et gruppenavn og der kan så tilføjes de antal venner man ønsker.</li> </ol> </li> <li>2. Der trykkes på opret gruppe</li> <li>3. Database tabellen groups bliver opdateret, med at der nu er en ny ejer af en gruppe</li> <li>4. Database tabellen group_members bliver opdateret med de nye medlemmer</li> </ol>
<b>Post conditions:</b>
1. Gruppe oprettes
<b>Alternative flows:</b>
<ol style="list-style-type: none"> <li>1. Programmet crasher</li> <li>2. Databasen går ned</li> <li>3. Intet internet</li> </ol>

<b>Use Case: Log ud</b>
ID: 6
<b>Brief Description:</b> Bruger ønsker at tage en pause fra programmet og logge ud.

<b>Primary actors:</b>
1 Bruger
<b>Secondary actors:</b>
<b>Preconditions:</b>
<ol style="list-style-type: none"> <li>1. Use Case 2 er opfyldt.</li> </ol>
<b>Main flow:</b>
<ol style="list-style-type: none"> <li>1. Brugeren trykker på indstillinger og derefter log ud</li> <li>2. Brugeren sættes til online=0 i databasen</li> </ol>
<b>Post conditions:</b>
<ol style="list-style-type: none"> <li>1. Brugeren er nu logget ud og kan ikke bruge programmets funktioner</li> </ol>
<b>Alternative flows:</b>
<ol style="list-style-type: none"> <li>1. Programmet crasher</li> <li>2. Databasen går ned</li> <li>3. Intet internet</li> </ol>

<b>Use Case:</b> Deaktiver bruger
<b>ID:</b> 7
<b>Brief Description:</b>
Bruger ønsker ikke længere at bruge programmet og ønsker at deaktivere sin bruger.
<b>Primary actors:</b>
Bruger 1
<b>Secondary actors:</b>
<b>Preconditions:</b>
<ol style="list-style-type: none"> <li>1. Use Case 2 er opfyldt.</li> </ol>

**Main flow:**

1. Der klikkes på Min Profil og derefter indstillinger
  - a. Der trykkes derefter på deaktivere bruger, hvor der bedes om at indtaste et password
  - b. Brugeren trykker på knappen deaktivere
2. Databasen opdateres
  - a. Brugeren bliver sat til deleted=1
  - b. Brugeren bliver sat til at være offline

**Post conditions:**

1. Brugeren har ikke længere adgang til programmet.

**Alternative flows:**

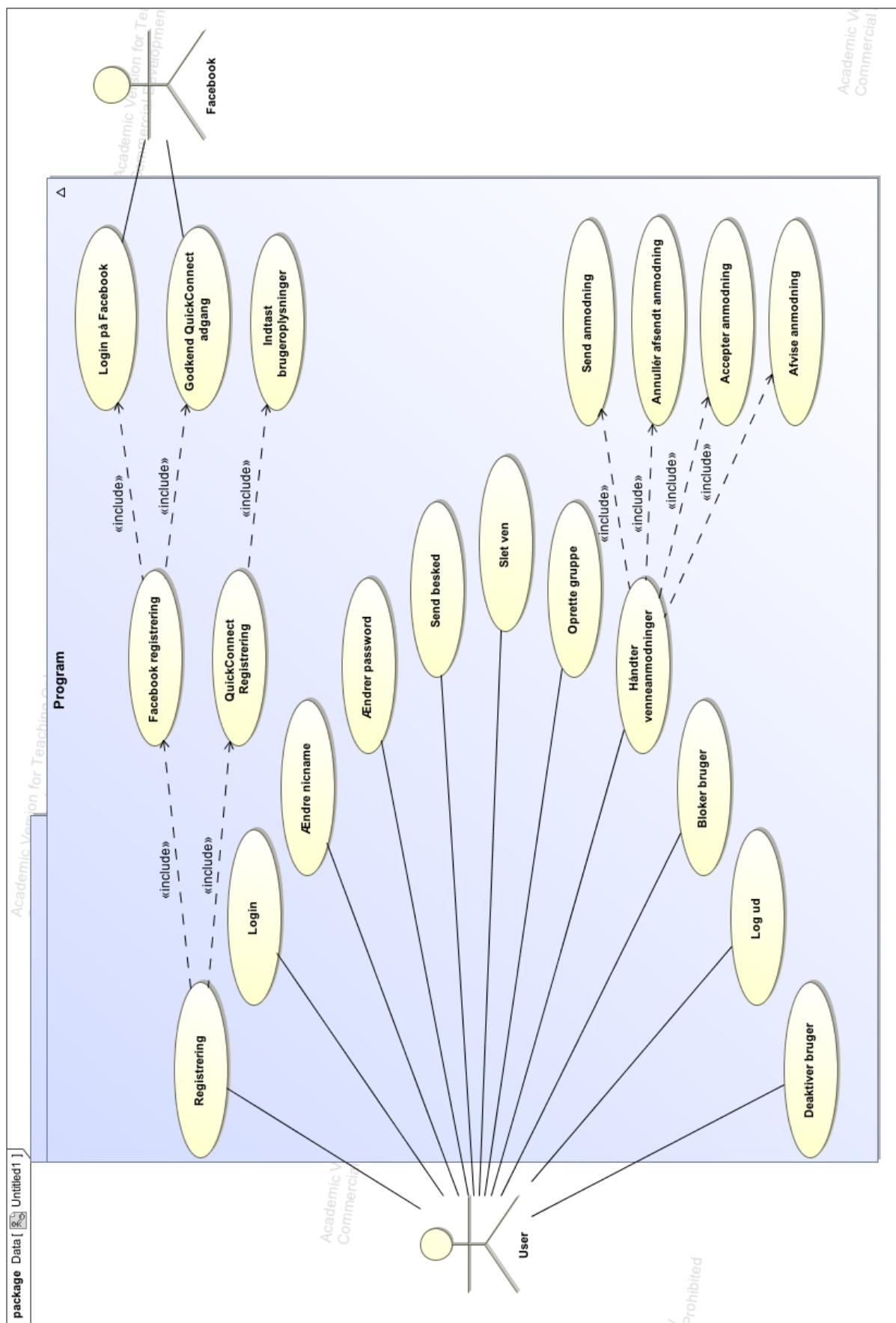
1. Programmet crasher
2. Databasen går ned
3. Intet internet

### **3.3. Use case diagram**

Ud fra de foregående use case beskrivelser har vi lavet følgende use case – diagram, som illustrerer systemet ret godt. Det er en del low level use cases som vi har valgt fra, da de er for simple og ikke har den store betydning for udformning af programmet.

Som det ses har vi i dette diagram 5 high-level use cases for spiller. Ud fra to af disse use cases har vi uddybet yderligere, hvad systemet skal kunne gøre.

Vi har desuden inddraget Facebook som værende aktør, da man ved Facebook registrering naturligvis skal igennem Facebooks API og deres godkendelse.



### **3.4. BCE model**

Vi har vores 2 aktører i vores chatprogram. Først og fremmest har vi udviklerne, som er os. Derudover har vi brugerne af vores program. I vores chatprogram skal man have mindst 1 ven, for at kunne skrive med en anden selvfølgelig.

Den første type, som fremgår i BCE modellen, er Boundary, hvilken vi har 2 af i vores program. Disse er nemlig GUI og TUI. Det er her, at det hele bliver startet op. GUI'en er det grafiske vindue, som brugeren får vist, når programmet startes op. Brugeren af programmet har ikke adgang til andet end dette. TUI'en er det tekstdbaserede, altså koden. Det har kun udvikleren adgang til, og selvfølgelig har vi også adgang til selve programmet.

Vi har 3 typer af Control. Det ene er vores main klasse, som sørger for at starte det, det andet

er controller klassen, som får besked på at starte loginwindow, og det tredje er test, hvor vi skal teste de forskellige dele i vores kode. Controller starte loginwindow klassen, og sørger derved for at det hele bliver startet op.

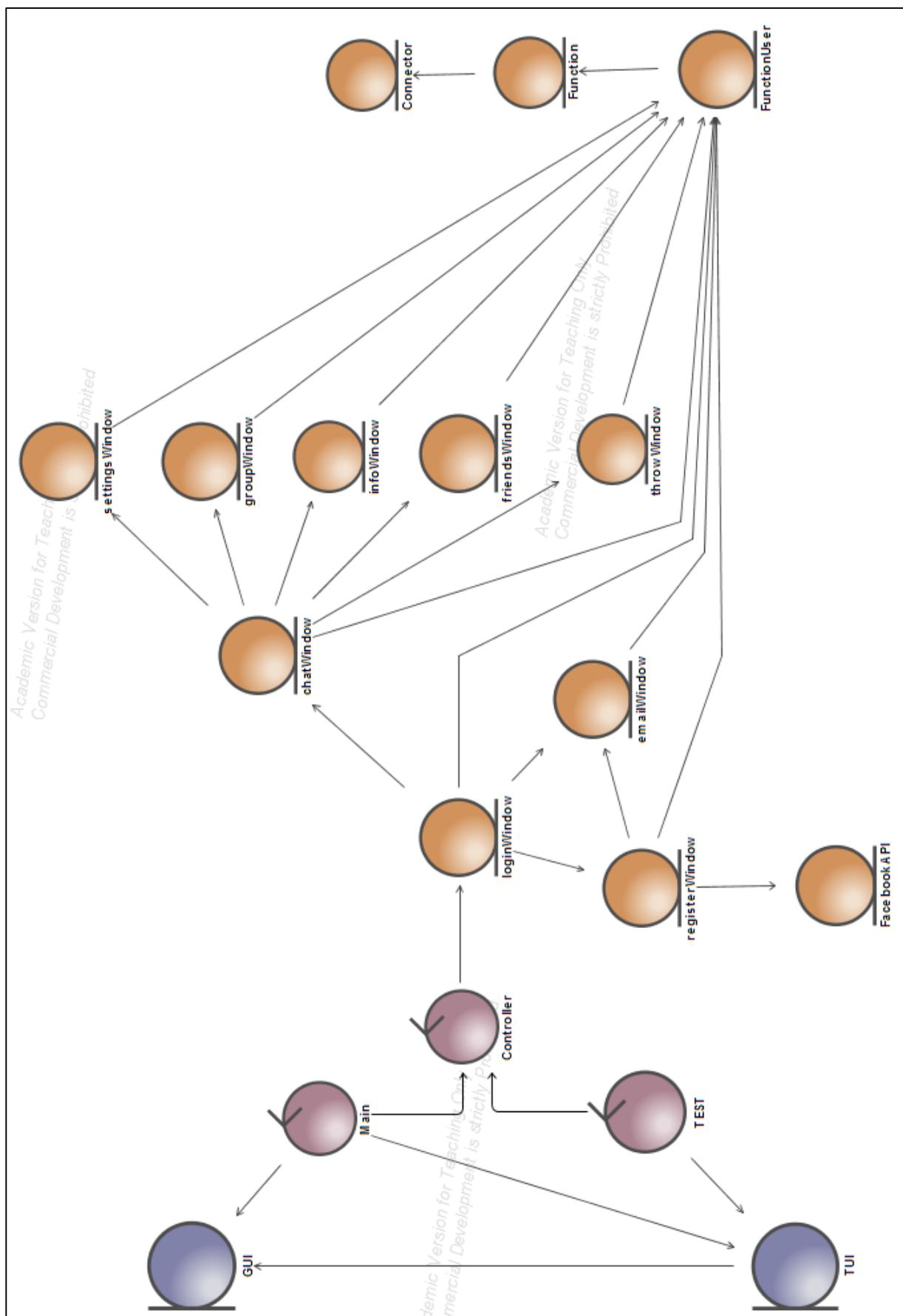
Til sidst har vi vores Entity typer, som vi faktisk har en del af, da det er blevet et større program, hvor man, som bruger har mange muligheder. Vi har følgende Entity klasser:

- loginWindow
- chatWindow
- registerWindow
- emailWindow
- FacebookAPI
- settingsWindow
- groupWindow
- infoWindow
- friendsWindow
- throwWindow
- FunctionUser
- Function
- Connector

Det er her faktisk loginWindow, som står for at starte hele programmet op. I vores controller klasse, sker der ikke rigtig andet, end at starte loginWindow op, som så starter programmet. Hernæst kan man så komme ind på enten chatWindow, registerWindow eller emailWindow, alt efter hvad man vil. Når man så kommer ind på en af de klasser, har man mulighed for forskellige valg igen. Man kan eksempelvis komme ind på settingswindow fra chatWindow, og foretage noget ønsket derinde. På denne måde har vi Entity klasser

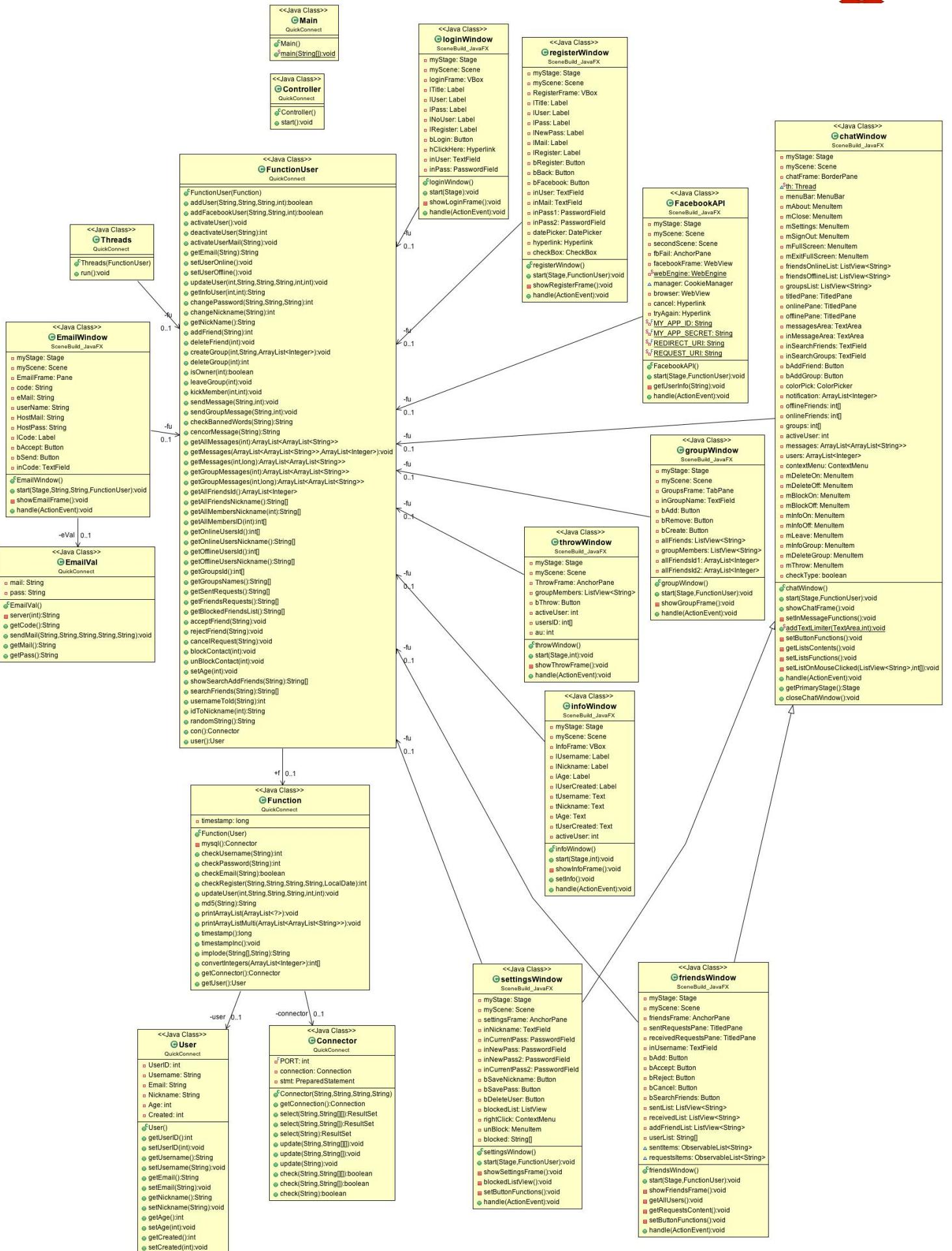
, som egentlig bare kan køre i ring, og man kan nogen forskellige ting. Dog er det værd at bemærke, at vi har en klasse, som hedder FunctionUser, en klasse, som hedder Function, og en som hedder Connector. De fleste "Window"-klasser kalder FunctionUser klassen, hvor de fleste af vores metoder står. Metoderne herinde sørger for, at brugerne kan foretage de ønskede ting i programmet, det er altså for brugeren. Function klassen sørger for, at komme med de nødvendige funktioner, som FunctionUser skal bruge. Til sidst har vi Connector klassen, som sørger for at forbinde det til vores database. Dette kommer vi også mere detaljeret ind på, lidt længere fremme i rapporten.

På denne måde har alle de forskellige Entity klasser adgang til de forskellige metoder i FunctionUser, og mulighed for at checke, opdatere eller slette nødvendige informationer i vores database.



### 3.5. Klasse diagram

På følgende side ses vores klassediagram. Dette diagram indeholder rigtig mange ting og derfor er det ikke manuelt lavet men genereret automatisk i Eclipse.



## 4. Implementering

### 4.1. Forbindelse til databasen

#### Hvordan forbinder vi til databasen?

Vi opretter forbindelsen til databasen igennem JDBC Driver og forbinder igennem url'en som opretter forbindelsen til databasen med følgende variabler: HOST, DATABASE, USERNAME & PASSWORD.

```

12     private final int PORT = 3306;
13     private Connection connection;
14     public Connector(String HOST, String DB, String UN, String PW) {
15         try {
16             Class.forName("com.mysql.jdbc.Driver");
17             String url = "jdbc:mysql://" + HOST + ":" + PORT + "/" + DB;
18             connection = DriverManager.getConnection(url, UN, PW);
19         } catch(ClassNotFoundException | SQLException e) {
20             e.printStackTrace();
21             System.exit(1);
22         }
23     }

```

Connector.java

Da disse er fortrolige oplysninger, og vi ønsker at de ikke skal findes på vores online repository ( [github.com](https://github.com) ) , har vi valgt at læse dem igennem en fil der hedder db.txt.

Filen skal ligges på roden af projektmappen før der kan forbindes til databasen.

Dette er naturligvis kun under udviklingsperioden men når programmet er færdiggjort og skal konverteres til en executable fil, så skal disse informationer ligge i koden.

```

27  private Connector mysql() {
28      BufferedReader br = null;
29      try {
30          String host, db, un, pw;
31          br = new BufferedReader(new FileReader("db.txt"));
32          host = br.readLine();
33          db = br.readLine();
34          un = br.readLine();
35          pw = br.readLine();
36          return new Connector(host, db, un, pw);
37      } catch(IOException e) {
38          e.printStackTrace();
39      } finally {
40          try {
41              if(br != null)
42                  br.close();
43          } catch(IOException ex) {
44              ex.printStackTrace();
45          }
46      }
47      return null;
48  }

```

Function.java

Vi har brugt Java Bufferreader klassen som vi har importeret i vores function.java. Denne klassen følger med i JAVA Platform SE 7. Her henter vi fra filen db.txt som skal læses og navngiver den typen BufferedReader br. Herefter læser vi ved metoden readLine(); hver linje for at definerer vi vores String host, db, un og password.

Indholdet af filen db.txt ser således ud:

```
mysql|13.gigahost.dk
ahma0942_dtu_livechat
ahma0942
_r0ot_s*d3v
```

Hvordan opdaterer og indsættes der i databasen?

Igennem vores projekt har vi valgt at bruge prepared statements til data manipulation.

For at sikre os at manipulationerne sker flydende og sikkert.

Hvad angår update metoderne har vi valgt at gøre brug af method overloading, afhængig af hvilke parameter man vælger at bruge.

```
56  public void update(String query, String[]... val) throws SQLException {
57      stmt = connection.prepareStatement(query);
58      for(int i = 1; i <= val.length; i++) {
59          if(val[i - 1][0].equalsIgnoreCase("l"))
60              stmt.setLong(i, Long.parseLong(val[i - 1][1]));
61          else if(val[i - 1][0].equalsIgnoreCase("i"))
62              stmt.setInt(i, Integer.parseInt(val[i - 1][1]));
63          else stmt.setString(i, val[i - 1][1]);
64      }
65      stmt.executeUpdate();
66  }
```

Function.java

I ovenstående metode har vi en metode som tager imod en String query som fx kunne se således ud: "INSERT INTO users Values (0,?, ?, ?, ?, ?, 0, ?, 0, 0, 0)".

her vil de efterfølgende parameter være String Arrays, hvor val[0] index sættes til enten være int ("i"), Long("l") eller String("") afhængig af hvad man indtaster i String array. Man kunne fx indsætte String [] ... val som vi vil have indsat i queryen. som følgende:

```

new String[] { "s", f.md5(pass) },
new String[] { "s", email },
new String[] { "i", "" + age },
new String[] { "l", Long.toString(f.timestamp()) }

```

Så vil den indsætte en tuple i tabellen users med et krypteret password, e-mail, alder, og timestamp.

### Hvordan henter vi data fra databasen?

For at hente data fra databasen har vi også method overloading ligesom de andre data manipulations metoder som vi har i vores connector klasse.

```

29  public ResultSet select(String query, String[]... val) throws SQLException {
30      stmt = connection.prepareStatement(query);
31      for(int i = 1; i <= val.length; i++) {
32          if(val[i - 1][0].equalsIgnoreCase("l"))
33              stmt.setLong(i, Long.parseLong(val[i - 1][1]));
34          else if(val[i - 1][0].equalsIgnoreCase("i"))
35              stmt.setInt(i, Integer.parseInt(val[i - 1][1]));
36          else stmt.setString(i, val[i - 1][1]);
37      }
38      ResultSet res = stmt.executeQuery();
39      return res;
40  }
41
42  public ResultSet select(String query, String... val) throws SQLException {
43      stmt = connection.prepareStatement(query);
44      for(int i = 1; i <= val.length; i++)
45          stmt.setString(i, val[i - 1]);
46      ResultSet res = stmt.executeQuery();
47      return res;
48  }

```

Connector.java

Hvis man vil hente data fra databasen så skal man kører en af de 3 select(...) metoder vi har, her returnerer den en ResultSet res med alle de tupler man nu vil returnere vha queryen. String [] val fungerer på samme måde som i update(...) metoden beskrevet foroven.

Vi har i vores brugt denne metode igen og igen for at returnerer online, offline brugere og til at vise blokerede venner osv.

### Check() metoderne

Vi har oprettet nogle check() metoder, så man fx kan tjekke om en bruger fx er ejeren af en bestemt gruppe. I dette eksempel bruger vi denne metode:

```

94  public boolean check(String query) throws SQLException {
95      ResultSet res = select(query);
96      if(res.next())
97          return true;
98      return false;
99  }

```

Connector.java

```
con().check( "SELECT owner_id FROM groups where owner_id = " + user().getUserID() + "
AND group_id= " + groupID)
```

Her vil denne metode returnere en boolean afhængig af om der dukker et resultat op i ResultSet res eller ej.

## 4.2. Mail

Vi har valgt at bruge en email validering i programmet, for at gøre vores program mere sikkert, og dermed sikre at ingen falske mails tages i brug.

Når brugeren vælger at registrere sig hos QuickConnect, sendes kode til pågældende brugers mail, som han skal indtaste. Hvis koden indtastes korrekt bliver han registeret og aktiveret som bruger i databasen.

Koden bliver genereret i Eclipse, og sendes fra vores server QuicConnect.tk hos gigahost.

```

42  public String getCode() {
43      int digit1 = (int) ((Math.random() * 10));
44      int digit2 = (int) ((Math.random() * 10));
45      int digit3 = (int) ((Math.random() * 10));
46      int digit4 = (int) ((Math.random() * 10));
47      String code = "" + digit1 + "" + digit2 + "" + digit3 + "" + digit4;
48      return code;
49  }

```

EmailVal.java

Klassen læser mail og password fra mail.txt filen, og den første linje af filen bliver gemt som hostmail, og den anden del bliver gemt som hostpassword.

```

21  private String server(int i) {
22      BufferedReader br = null;
23      try {
24          br = new BufferedReader(new FileReader("mail.txt"));
25          mail = br.readLine();
26          pass = br.readLine();
27      } catch(IOException e) {
28          e.printStackTrace();
29      } finally {
30          try {
31              if(br != null)
32                  br.close();
33          } catch(IOException ex) {
34              ex.printStackTrace();
35          }
36      }
37      if(i == 0)
38          return mail;
39      else return pass;
40  }

```

EmailVal.java

I klassen ligger der også en sendMail metode, som sender mail til brugerne.

Til at starte med, aktiverer vi STARTTLS, til at sikre forbindelse. Hosten er smtp.gigahost.dk, som vores hjemmeside ligger på. Dermed bruger vi hostmail og hostpassword, som allerede er defineret længere oppe i koden. Vi bruger portnummer 587, og derefter godkender vi adgang til serveren. På den måde kan der sendes en mail via programmet, til de forskellige brugere.

```
51④ public void sendMail(String from, String pass, String to, String subject, String body) {  
52     Properties props = System.getProperties();  
53     String host = "smtp.gmail.com";  
54     props.put("mail.smtp.starttls.enable", "true");  
55     props.put("mail.smtp.host", host);  
56     props.put("mail.smtp.user", from);  
57     props.put("mail.smtp.password", pass);  
58     props.put("mail.smtp.port", "587");  
59     props.put("mail.smtp.auth", "true");
```

EmailVal.java

### 4.3. Facebook

En vigtig del af vores projekt var at vi skulle gøre det muligt for slutbrugeren at validere sig igennem Facebook eller Google.

Det tilføjer nemlig en hel anden dimension til programmet.

Vi gjorde vores bedste for at indbygge både Facebook og Google validering, men da vi kunne se at tiden var knap, kunne vi kun nå en af dem.

Det lykkedes os at få Facebook API'en til at fungere.

Det første vi gjorde var at oprette en app hos Facebook. Hvor vi så fik tildelt et App ID og en App hemmelighed. Dette gjorde det muligt for os at spørge facebook brugeren om deres offentlige oplysninger, deres email samt deres venner.

Herefter skulle dette inkorporeres i vores kode. Vi lavede en klasse som hedder FacebookAPI, som forbindet til Facebook via en webView.

Der er en webEngine som sender en request for at åbne Facebook login dialogen. Herefter sættes en listener på som lytter efter ændringer i URL.

```

82     webEngine = browser.getEngine();
83     webEngine.getLoadWorker().stateProperty().addListener(new ChangeListener<State>() {
84         @Override
85         public void changed(@SuppressWarnings("rawtypes") ObservableValue ov, State oldState, State newState) {
86             if(newState == Worker.State.SUCCEEDED) {
87                 String url = webEngine.getLocation();
88                 myStage.setTitle(url);
89                 if(url.contains("#access_token")) {
90                     String accessUrl = webEngine.getLocation();
91                     String token = accessUrl.substring(accessUrl.indexOf("#access_token") + 1);
92                     getUserInfo(token);
93                 }
94                 if(url.contains("error_code=200")) {
95                     myStage.setScene(secondScene);
96                 }
97             }
98         }
99     });
100    webEngine.load(REQUEST_URI);

```

Når en access token bliver givet efter login og tilladelse fra brugeren så kan denne aflæses og oplysninger kan hentes.

```
108 private void getUserInfo(String token) {  
109     AccessToken tokenInfo = AccessToken.fromQueryString(token);  
110     FacebookClient fbClient = new DefaultFacebookClient(tokenInfo.getAccessToken(), MY_APP_SECRET,  
111         Version.VERSION_2_0);  
112     User user = fbClient.fetchObject("me", User.class,  
113         Parameter.with("fields", "name, first_name, last_name, email, age_range, gender"));  
114 }
```

#### 4.4. Vigtige ting i forhold til koden

##### Hvordan bliver en bruger sat offline?

Når en bruger lukker sit program, eksempelvis ved at trykke på Alt+F4, vil programmet ikke have nogen anelse at den er lukket, når den lukkes.

Derfor bruger vi en smart metode til at opdatere om en er online eller offline, og det er ved at opdatere hans online-status fra en af hans venner.

Det gøres ved at bruge feltet 'last\_on', ved at tjekke om den har været opdateret i de seneste 10 sekunder. Hvis den ikke har været opdateret i de seneste 10 sekunder, vil han sætte brugeren til at være offline. Hvis brugeren ikke har nogle venner og går offline, vil han i databasen stadig stå som online, men det vil ikke påvirke nogen eller noget, da der ikke er nogen grund til at sætte ham som offline, hvis ingen er der til at se om han er online eller offline.

##### Hvorfor hentes timestampen fra internettet og opdateres i koden hvert sekund, istedet for at hente den fra computeren?

Clocken på ens computer er det der bestemmer hvad ens timestamp er.

Clocksne på ens computer kan enten kan være beskadiget og eller forkerte.

Man kan også selv ændre på dem. Derfor henter vi timestamp fra en server og opdaterer timestampen hvert sekund med funktionen timestampInc().

På den måde har alle brugere samme timetamp når de logger ind.

## 4.5. Threads

I vores program, har vi 2 Threads der kører. Den første kører i Threads-klassen og ser således ud:

```

12@Override
13 public void run() {
14     while(true) {
15         try {
16             fu.f.timestampInc();
17             Thread.sleep(1000);
18         } catch(InterruptedException | IOException ex) {
19             Thread.currentThread().interrupt();
20         }
21     }
22 }
```

Threads.java

Denne Thread kører funktionen timestampInc() som ligger i funktion-klassen, hvert sekund.

Det giver mere mening at forklare om hvordan vi henter vores timestamp<sup>1</sup> før der forklares om hvad timestampInc() gør.

Vi startede med at hente timestamp fra selve computeren, men fandt hurtigt ud af at clocksne på forskellige computere, enten kan være beskadiget og ellers forkerte. Man kan også selv ændre på dem. Derfor henter vi timestamp fra en server<sup>2</sup>.

Det som timestampInc() funktionen gør er, at den inkrementerer timestampen hvert sekund, som hentes fra serveren første gang.

Så Threads-klassen står for at opdatere timestampen således at de er ens for alle brugere som er logget ind.

<sup>1</sup> Timestamp eller unix-timestamp er en måde at udvælge en specifik tid på, uden påvirkelse af tidszone. En timestamp er tiden fra starten af unix-epoken, indtil en specifik tid.

<sup>2</sup> Vi bruger <http://s8dev.org/timestamp.php> til at hente unix-timestamp.

Den anden Thread der køres, ser således ud:

```

85@     Task<Void> task = new Task<Void>() {
86@         @Override
87@         public Void call() throws Exception {
88@             while(true) {
89@                 Platform.runLater(new Runnable() {
90@                     @Override
91@                     public void run() {
92@                         try {
93@                             if(checkType == false) {
94@                                 ArrayList<ArrayList<String>> msg = fu.getGroupMessages(activeUser);
95@                                 for(int i = 1; i <= msg.get(0).size(); i++) {
96@                                     messagesArea.appendText(
97@                                         msg.get(1).get(i - 1) + ":\n" + msg.get(0).get(i - 1) + "\n\n");
98@                                     }
99@                             } else {
100@                                 fu.getMessages(messages, users);
101@                                 for(int i = 1; i <= messages.size(); i++) {
102@                                     for(int t = 1; t <= messages.get(i - 1).size(); t++) {
103@                                         if(users.get(i - 1) == activeUser)
104@                                             messagesArea.appendText(fu.idToNickname(users.get(i - 1)) + ":\n"
105@                                         + messages.get(i - 1).get(t - 1) + "\n\n");
106@                                         }
107@                                         if(users.get(i - 1) != activeUser) {
108@                                             notification.add(users.get(i - 1));
109@                                         }
110@                                         }
111@                                     }
112@                                     messages.clear();
113@                                     users.clear();
114@                                     getListsContents();
115@                                     fu.con().update("UPDATE users SET last_on='" + fu.f.timestamp()
116@                                         + "', online=1 WHERE user_ID='" + fu.user().getUserID() + "'");
117@                                 } catch(SQLException | IOException e) {
118@                                     e.printStackTrace();
119@                                 }
120@                             }
121@                         });
122@                         Thread.sleep(5000);
123@                     }
124@                 }
125@             };
126@     };

```

chatWindow.java

Denne Thread kører hvert 5'te sekund, og den ser om der er kommet nye beskeder fra andre brugere og, om den aktuelle bruger på chat-vinduet har sendt nye beskeder. Derudover, opdaterer den et felt i databasen som hedder 'last\_on', som fortæller hvornår brugeren sidst har været på.

## 5. Design-dokumentation

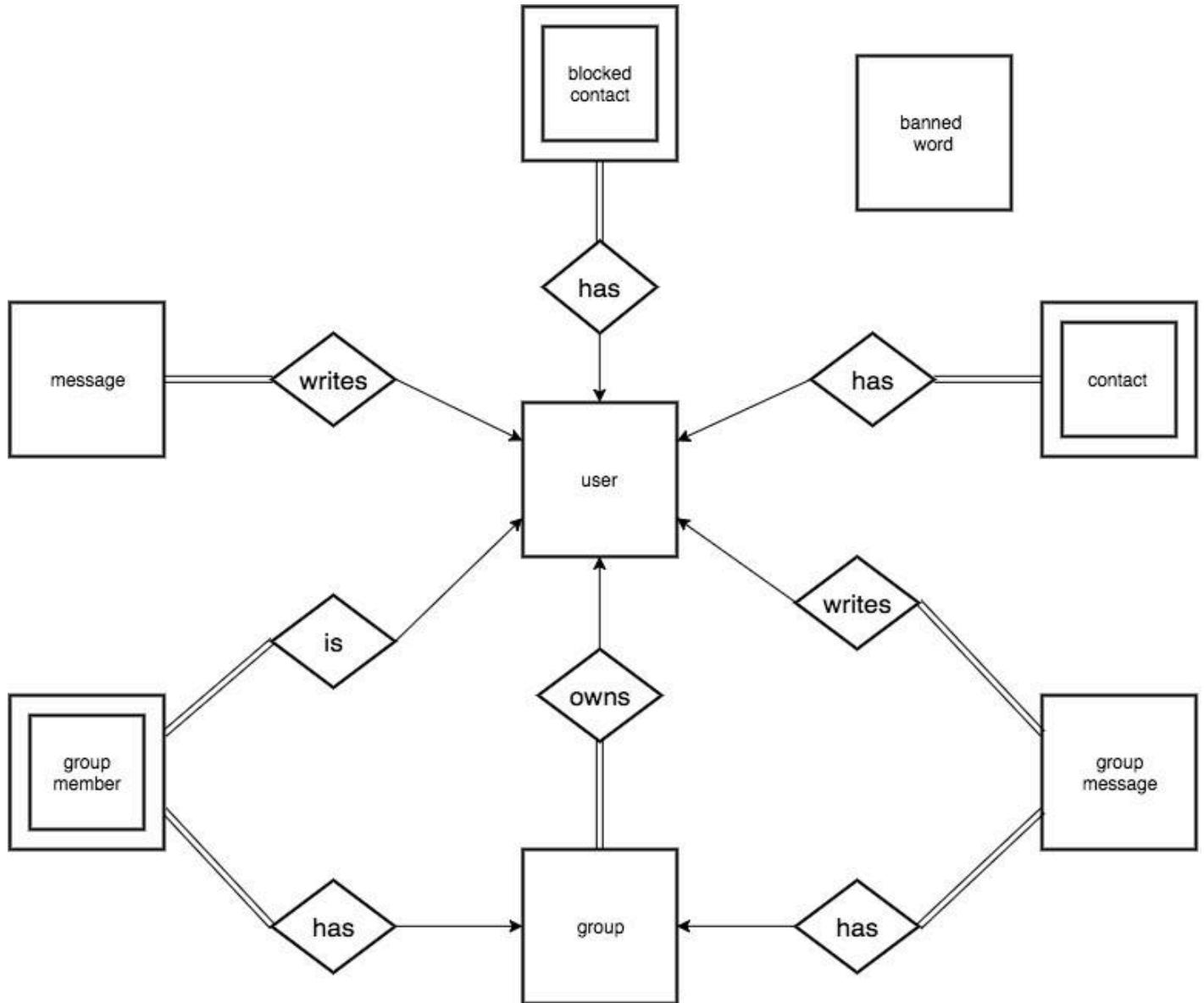
### 5.1. ER – diagram

Databasen er baseret på brugerne i programmet, og brugerne har forskellige funktioner som de kan udføre. En bruger kan have flere kontakter, og for at contact entity skal kunne eksistere i første omgang, skal man have en bruger entity, så det er en one-many relationship, og derfor er contact en weak entity. Det samme gælder for blocked contact, hvor man ikke kan blokere en bruger, uden at brugeren eksisterer. For at man kan have en gruppe medlem entity, skal man både være en bruger, og samtidig skal der eksisterer en gruppe, derfor der tale om many-one relationships, og derfor er det også en weak entity.

En gruppe besked, indeholder på samme måde også en bruger og kræver en gruppe. En bruger kan godt have gruppebeskeder, og en gruppe kan have mange beskeder, men en gruppebesked kan ikke blive sendt af flere end en bruger, eller til flere forskellige grupper. Derfor er der igen tale om en many-one relationship.

Vi har også en banned word entity, som ikke direkte har en forbindelse til de andre tabeller. I programmet bliver der tjekket om, en bruger sender en besked som indeholder bande-orde, derfor har den ikke en forbindelse til andre entities.

Følgende er en illustration af et ER – diagram som viser strukturen af QuickConnects database:



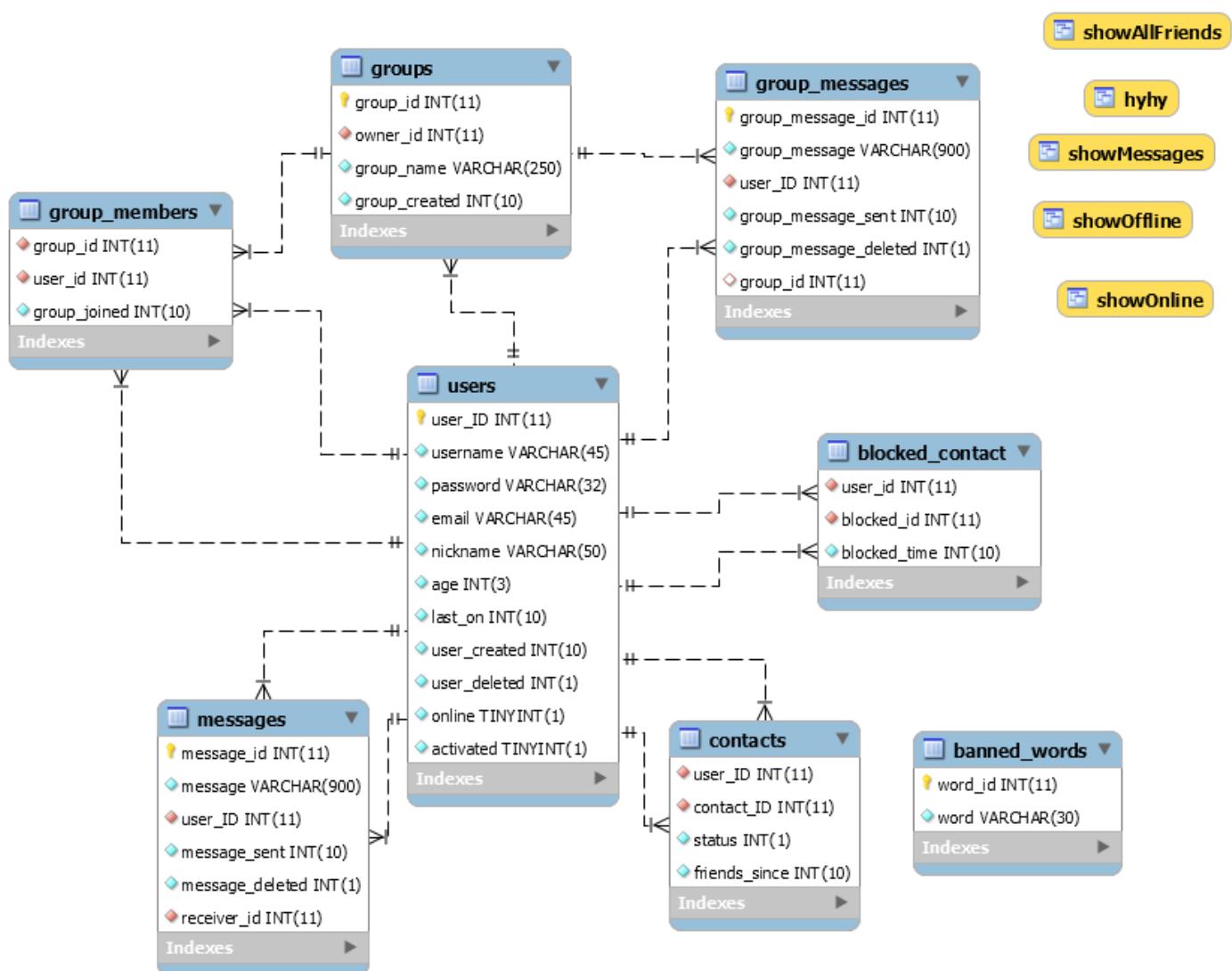
## 5.2. Schema diagram

Mens ER – diagrammet var brugbart og nyttigt før vi overhovedet lavet vores database, er følgende schema – diagram autogenereret ud fra vores database, efter at den var lavet. Diagrammet er især brugbart til at videreforsmilde og forklare strukturen af en given database.

Som der kan ses i schema – diagrammet, er alle tabeller afhængige af users tabellen, undtagen banned\_words. Det giver god mening, da man skal oprette en bruger i programmet, før man kan gøre noget som helst.

Diagrammet indeholder ligeledes 5 gule bokse, som repræsentere nogle views som er lavet i databasen, men disse er ikke relevante for dette projekt, da det ikke er blevet brugt i implementeringen.

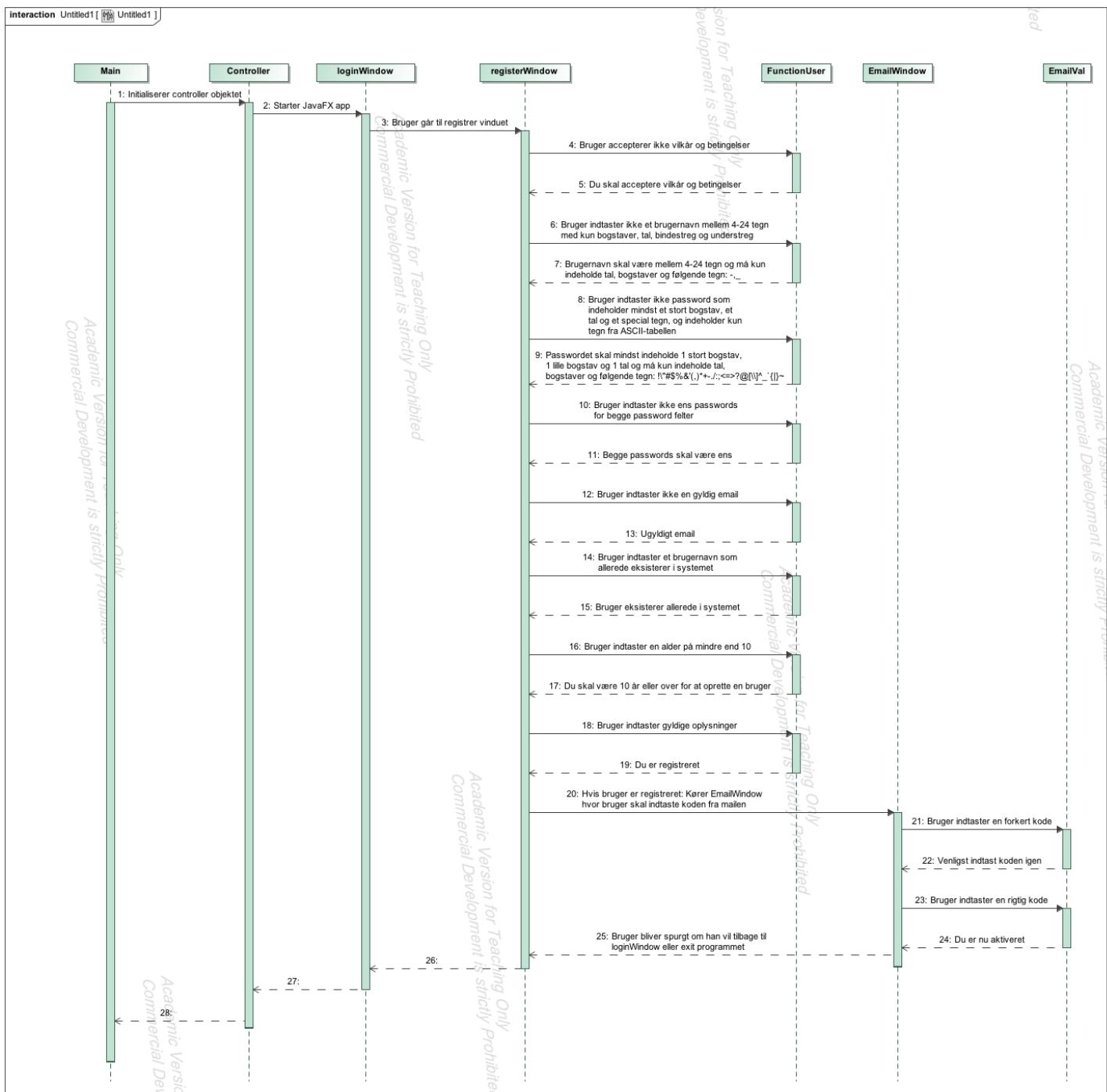
Nedenfor, ses schema – diagrammet for QuickConnects database. Dette diagram minder om det forrige ER – diagram men illustrerer dog tydeligere og mere klart hvordan databasen er opbygget og giver mere information om hver enkel tabel.



### 5.3. Design-sekvens diagram

Vi har valgt at skære design sekvens diagrammet i små bider, i det at diagrammet ellers vil blive alt for stort. Der er blevet valgt at udelade ting som er åbenlyse, og derfor kun taget de vigtige og mere kompliceret use cases, som vi så har lavet om til et design sekvens diagram. Alle disse diagrammer viser de processer der foregår, hver gang en bruger foretager en bestemt handling.

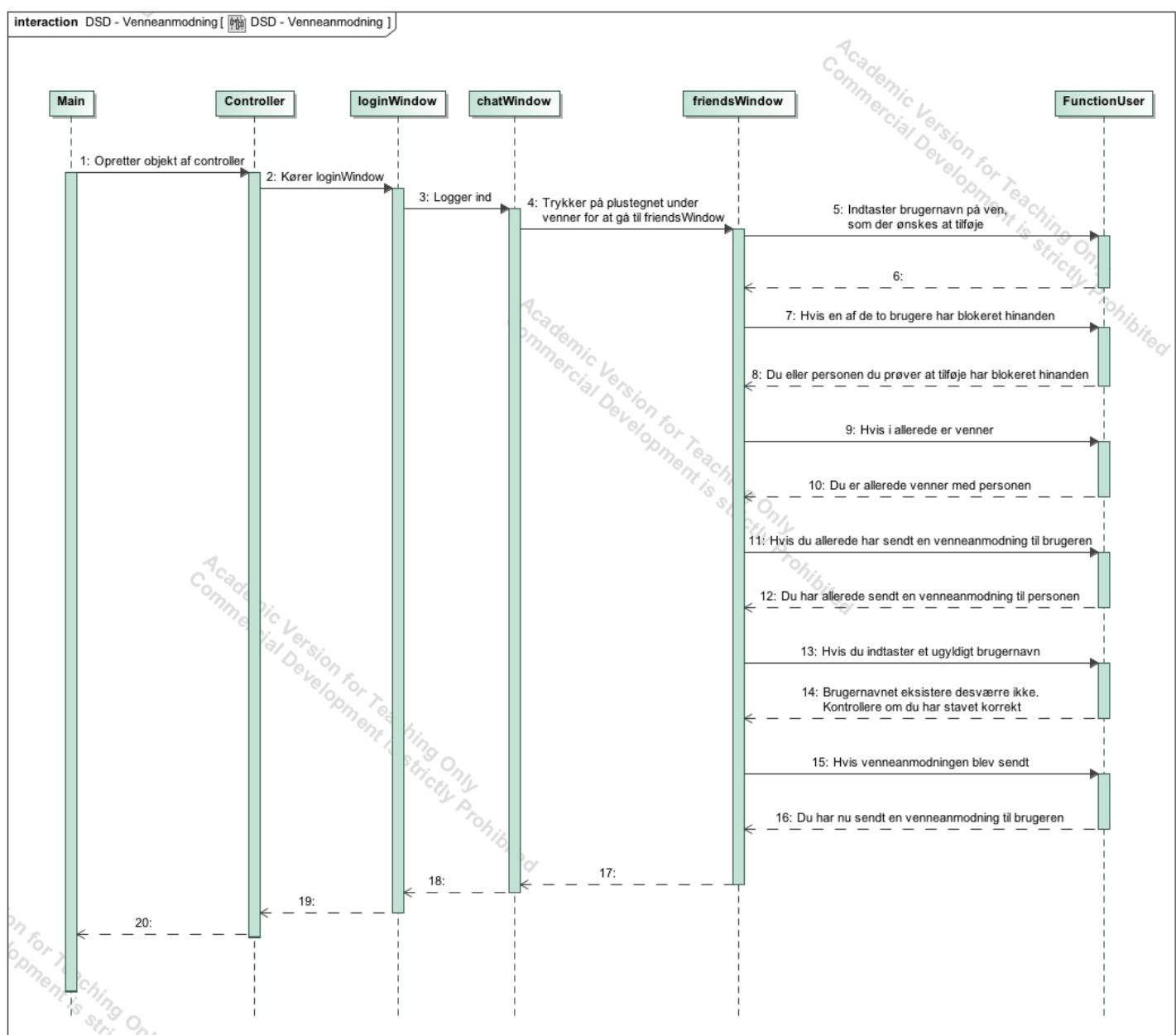
Nedenunder ses diagrammet over registrering.



Det kan ses at main kører Controlleren, som åbner loginWindow. Når der så klikkes på registrer, får brugeren to mulige måder at registrere sig på. Vi har valgt at vise QuickConnect registrering, i det er en mere kompliceret handling end facebook registrering.

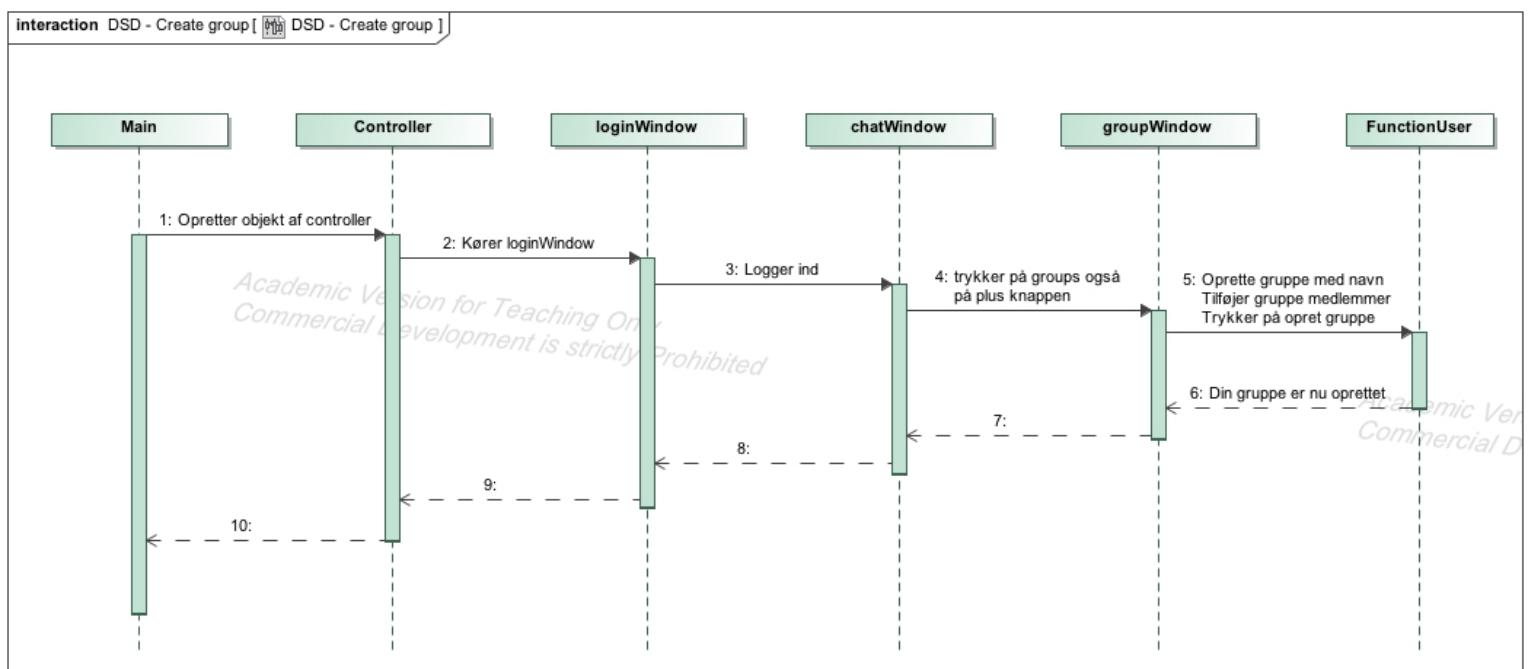
QuickConnect registrering har nogle forskellige betingelser, som kan ses overfor, før brugeren kan registrere sig.

Nedenunder ses et diagram over hvordan en **venneanmodning** bliver sendt.



Det interessante her, er at programmet først tjekker nogle forskellige ting før den sender en anmodning. Før anmodningen bliver sendt, tjekker programmet om brugeren er blokeret, om de allerede er venner, om en anmodning allerede er blevet sendt og om brugernavnet er ugyldigt. Hvis alle disse ting ikke gør sig gældende, ville venneanmodningen blive sendt.

Nedenunder ses et diagram over oprettelse af gruppe.

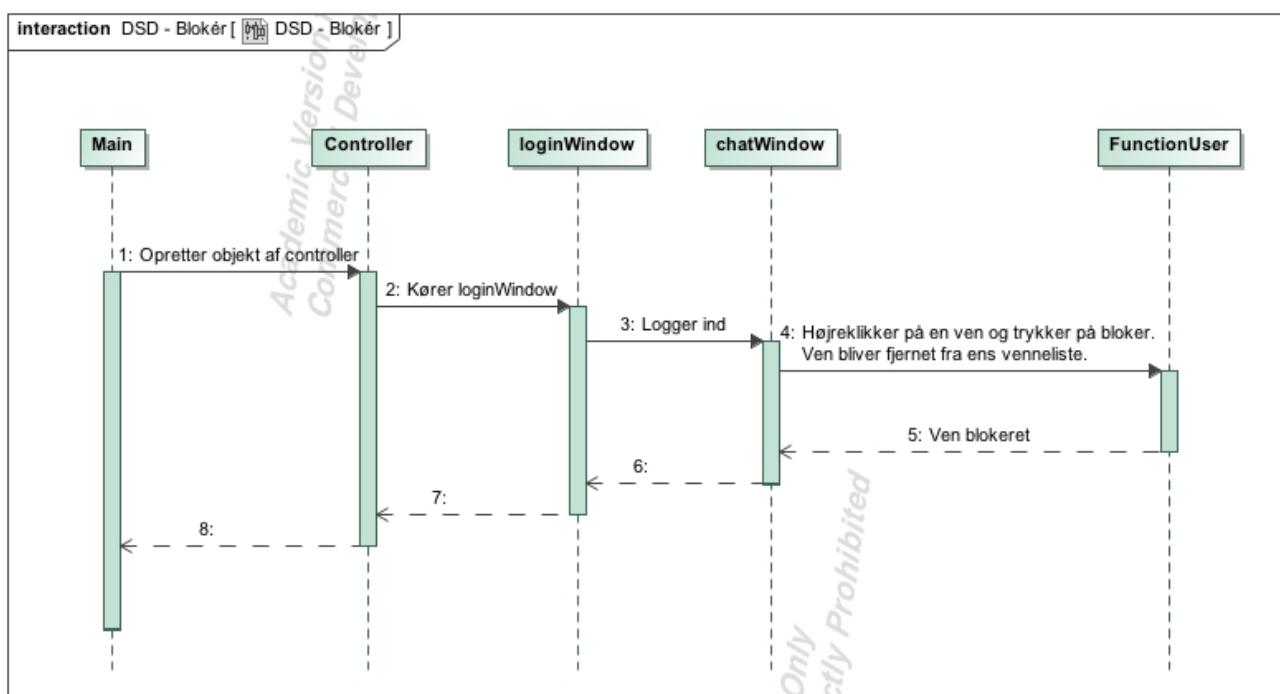


Dette diagram er et mere simpelt et af slagsen. Som det kan ses på det tidligere diagram, så kalder main altid controlleren, som derefter kører login vinduet. Når brugeren så logger ind, åbnes chatWindow. Brugeren klikker så på plustegnet i chatWindow åbnes gruppevinduet.

Brugeren indtaster et gruppnavn, og tilføjer de ønsket venner til gruppen.

Programmet svarer så tilbage at gruppen er oprettet, og den tilføjes derefter til databasen.

Nedenunder kan ligeledes ses hvordan en **bruger bliver blokeret**.



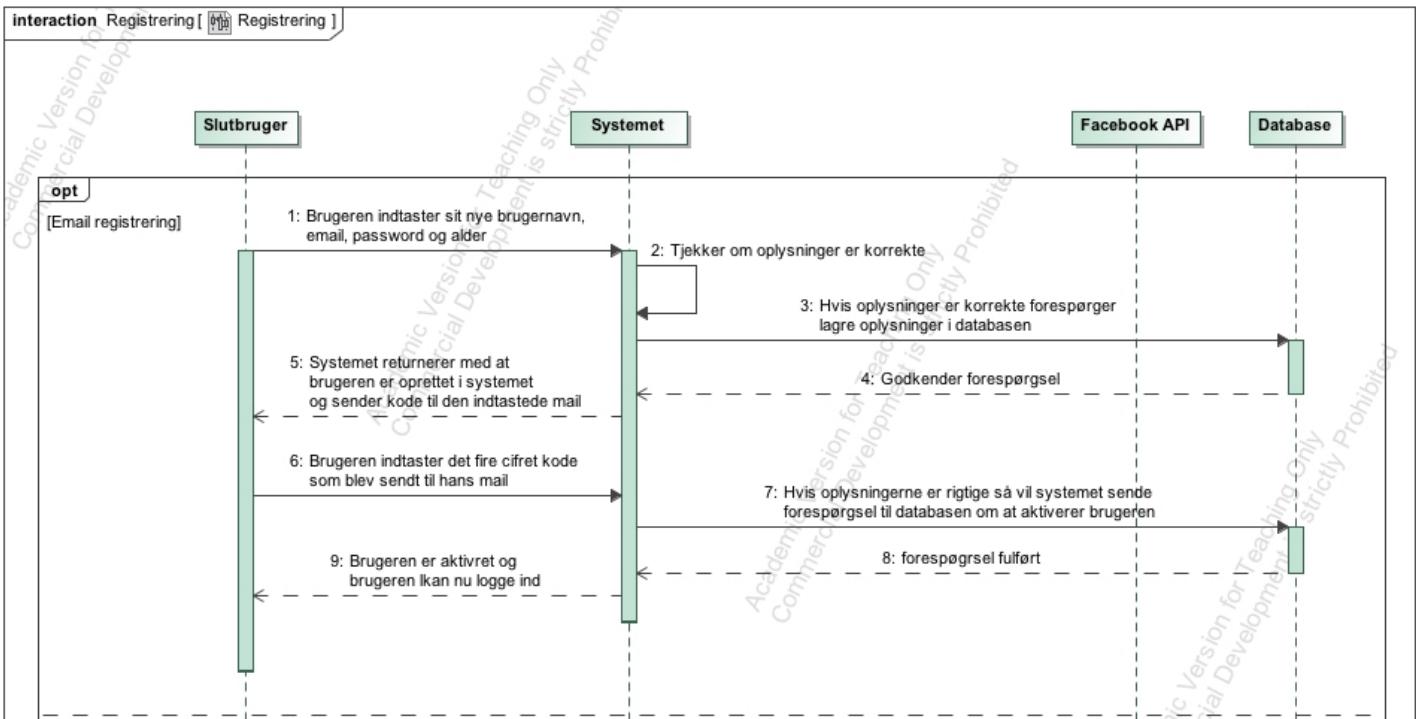
## 5.4. System-sekvens diagram

Vi har valgt at lave SSD'er for de vigtigste Use Cases vi har. Vi har følgende SSD'er:

### 5.4.1. Email registrering

Når brugeren vælger at trykke på registrer knappen så vil systemet åbne et vindue med de oplysninger som brugeren skal angive, såsom brugerens ønskede brugernavn, email password og alder. Så vil systemet tjekke om password og alder og brugernavn opfylder betingelserne. I det tilfælde hvor alle betingelser er opfyldt, så vil systemet forespørge databasen om at indsætte oplysningerne ind i databasen, som ikke-aktiveret. Og systemet giver besked til brugeren om at brugeren er oprettet.

Derefter sender systemet en mail til brugerens indtastede e-mail, som systemet beder brugeren om at indtaste. Brugeren indtaster koden, som systemet skal tjekke. Hvis dette er korrekt, vil systemet sende en forespørgsel til at aktiverer brugeren i databasen. Efter systemet har modtaget at denne forespørgsel er udført, så vil systemet give en besked til brugeren om at brugeren er nu aktiveret og brugeren kan nu logge ind med de registrerede oplysninger.

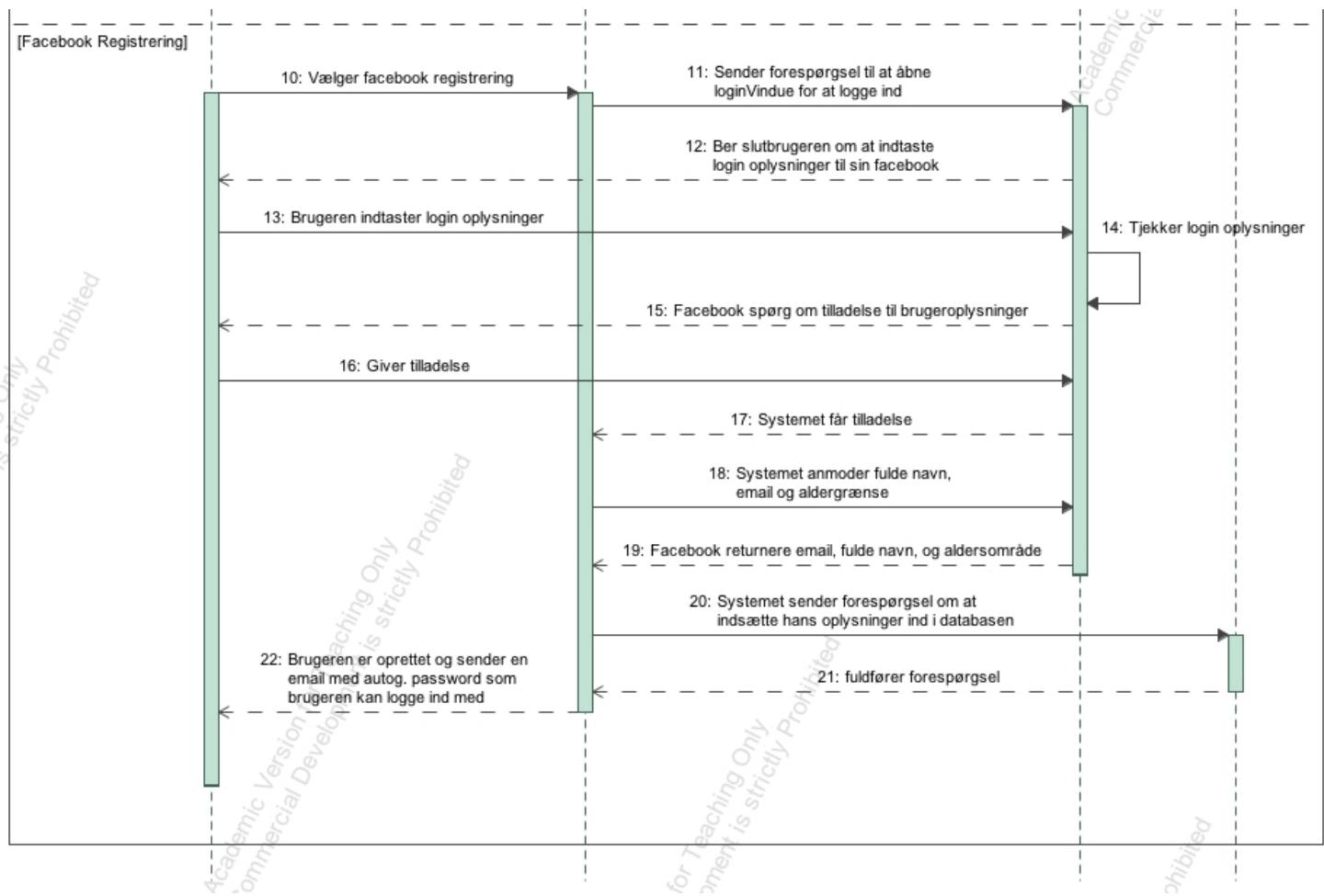


#### 5.4.2. Facebook registrering

Når brugeren vælger at trykke på registrér igennem Facebook knappen så vil systemet sende en forespørgsel til FACEBOOK API, om at åbne login vinduet til Facebook, herefter beder Facebook API, slutbrugeren om at indtaste login oplysninger til sin Facebook konto. Hvorefter FACEBOOK API, tjekker om de er korrekte. Facebook API returnerer med en besked om at der nu kan hentes oplysninger fra brugerens Facebook konto til systemet.

Systemet beder Facebook om at returnerer med fulde navn, e-mail, og aldersgrænse, hvorefter Facebook vil returnerer dem til systemet. Derefter sætter Systemet den registrerede brugers

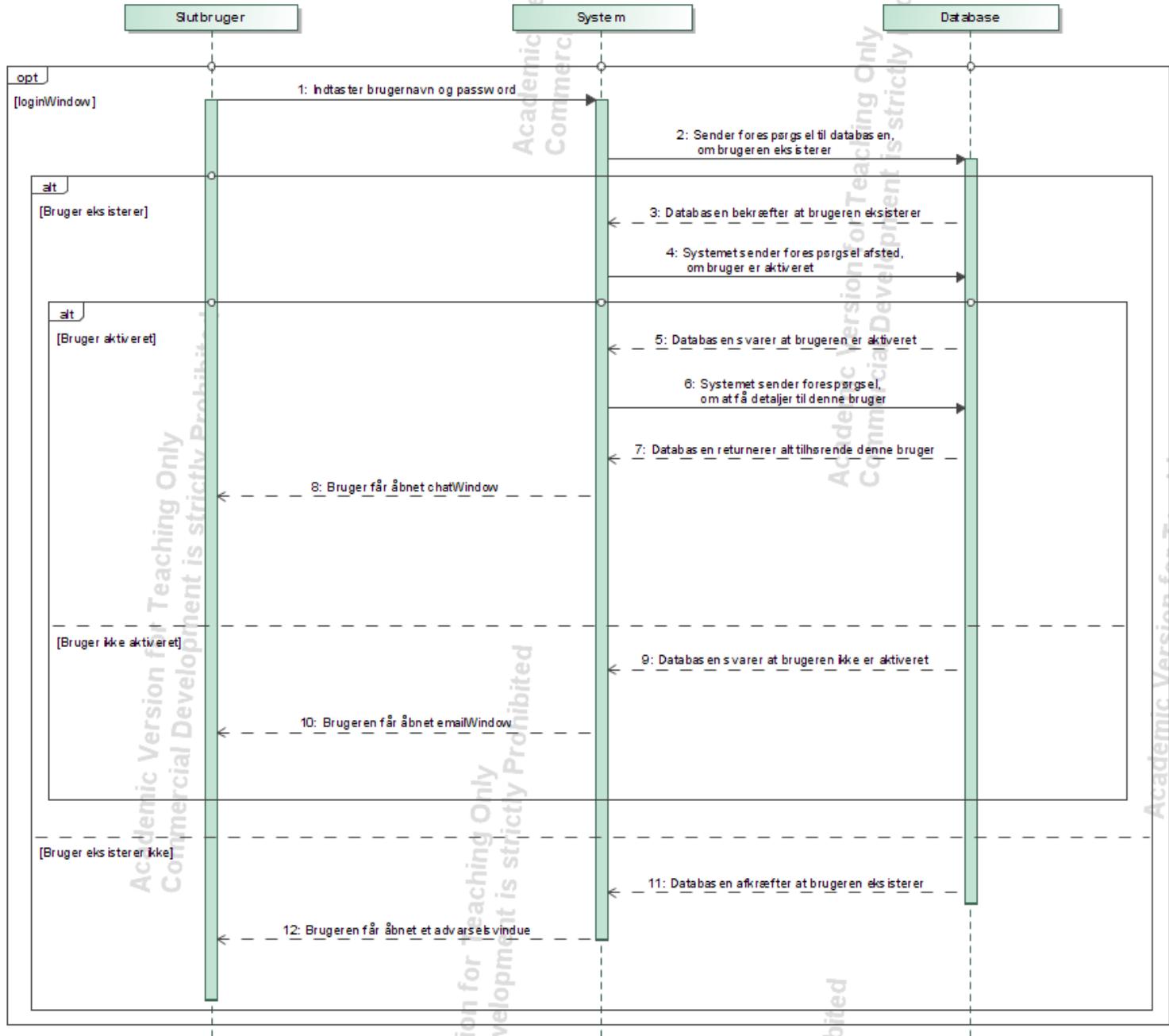
brugernavn til at være en e-mail, et autogenereret password, og e-mail som e-mail og sender forespørgsel til databasen om at indsætte dem ind i databasen. Når databasen returnere at det er gennemført, så vil systemet give besked til brugeren at brugeren er registrerede i systemet og kan nu logge ind med en password som er blevet sendt til hans mail.



### 5.4.3. Login

Her vil brugeren få mulighed for at indtaste brugernavn og adgangskode. Systemet vil scanne, det indtastede, og så sende forespørgsel til databasen, om hvorvidt det passer eller ej. Hvis brugeren eksisterer, vil systemet prøve at finde ud af, om brugeren er aktiveret, ved igen at spørge databasen. Hvis ikke brugeren er aktiveret, vil der blive spurgt, om en kode til at aktivere den. Hvis ikke brugeren eksisterer, vil der poppe et advarselsvindue op.

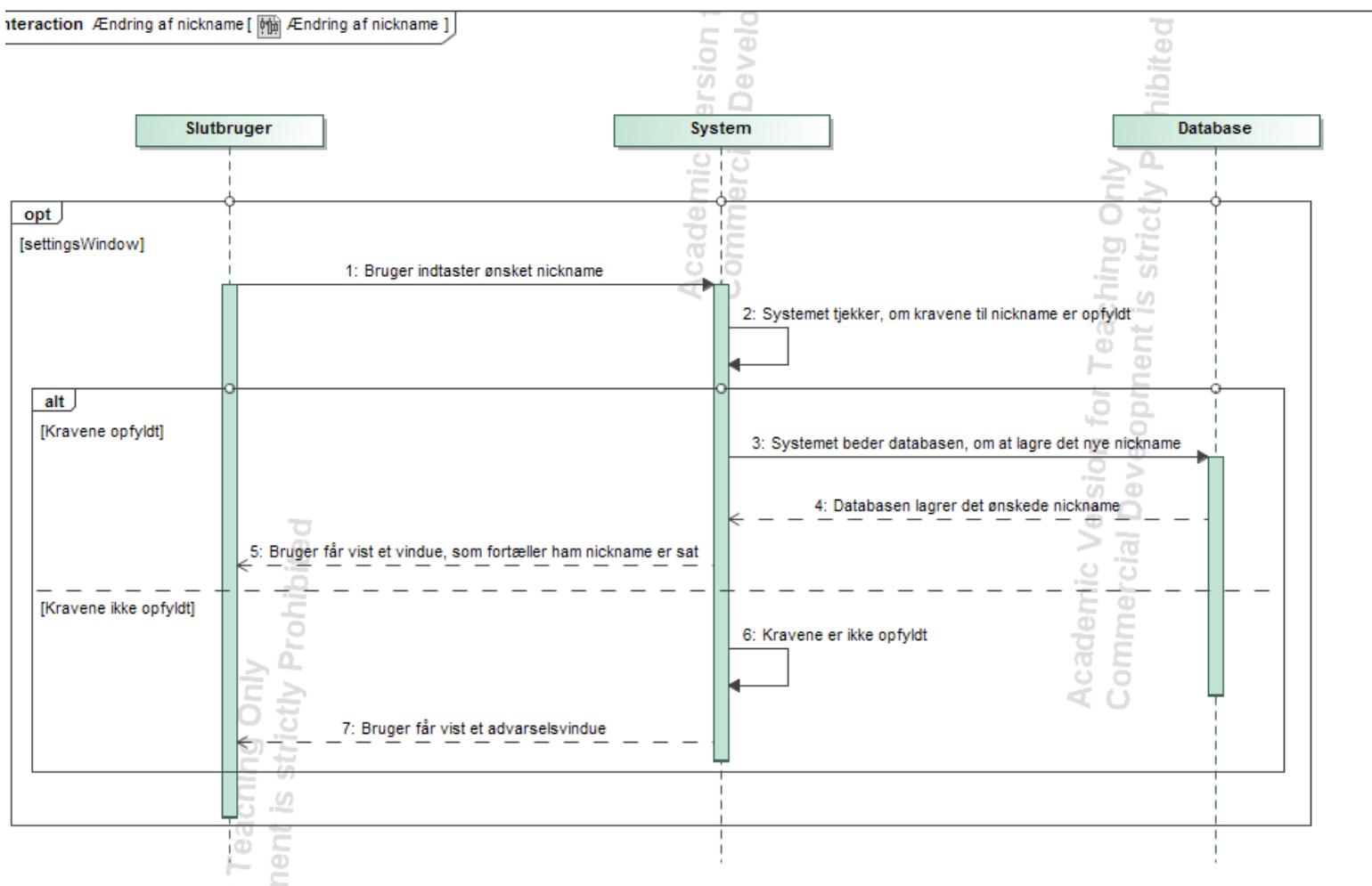
## Interaction Login [ Login ]



#### 5.4.4.

#### Ændring af nickname

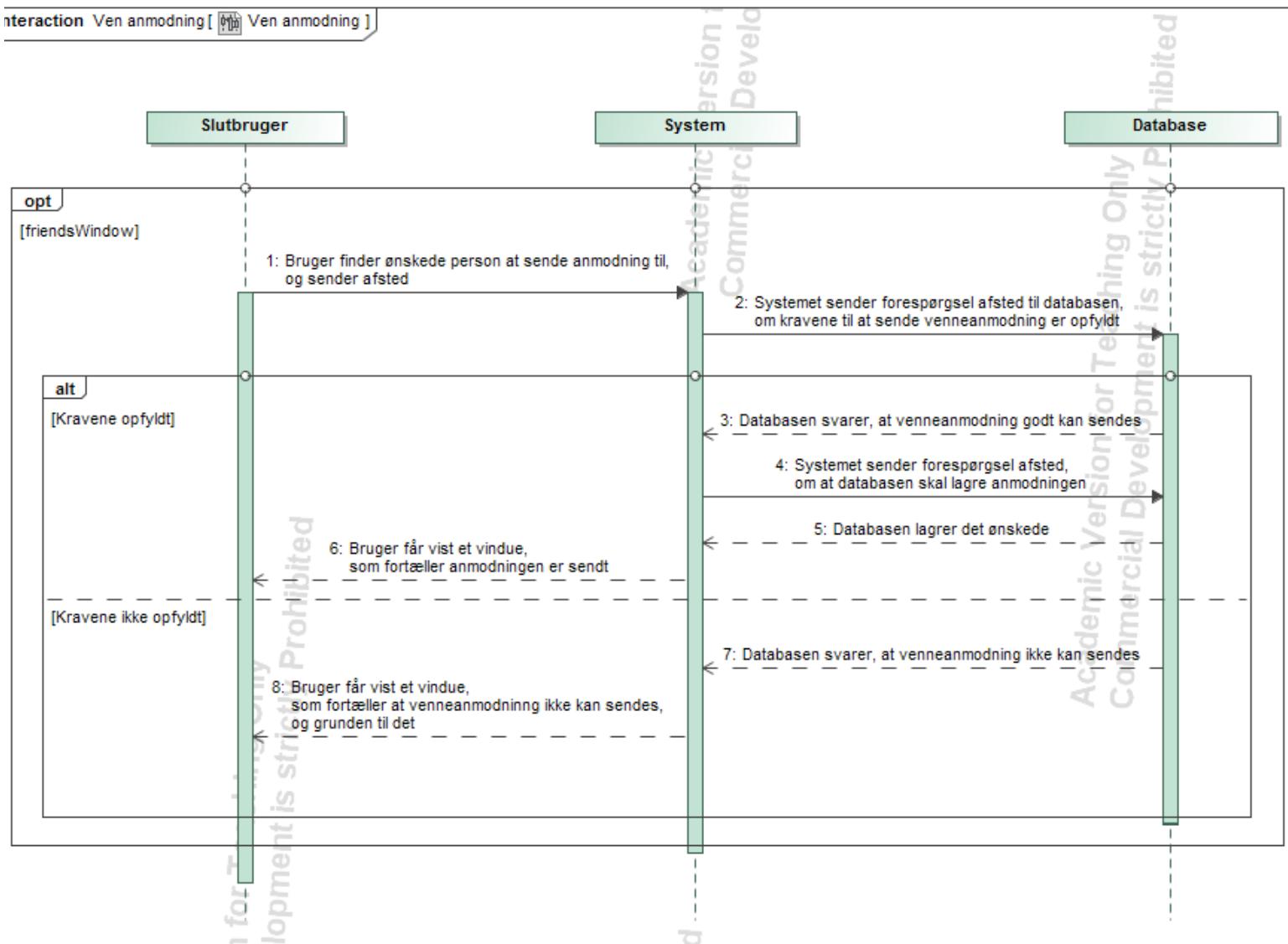
Her vil brugeren skulle gå ind på sine indstillinger, og indtaste det ønskede nickname. Derefter vil systemet tjekke, om de forskellige krav til det er opfyldt. Hvis kravene er opfyldt, vil databasen lagre det. Hvis ikke det er opfyldt, vil der poppe et advarselsvindue op.



#### 5.4.5.

#### Venneanmodning

Brugeren finder den ønskede person, at sende anmodning til, og sender det afsted. Systemet vil tjekke, om kravene er opfyldt for dette. Hvis kravene er opfyldt, vil databasen lagre anmodningen, og det vil blive vist, på den anden persons profil. Der popper så et vindue op, hvor personen får fortalt, at anmodningen er sendt. Hvis ikke kravene er opfyldt, vil brugeren få vist et pop-up vindue, med at anmodningen ikke kan sendes, og grunden til det.

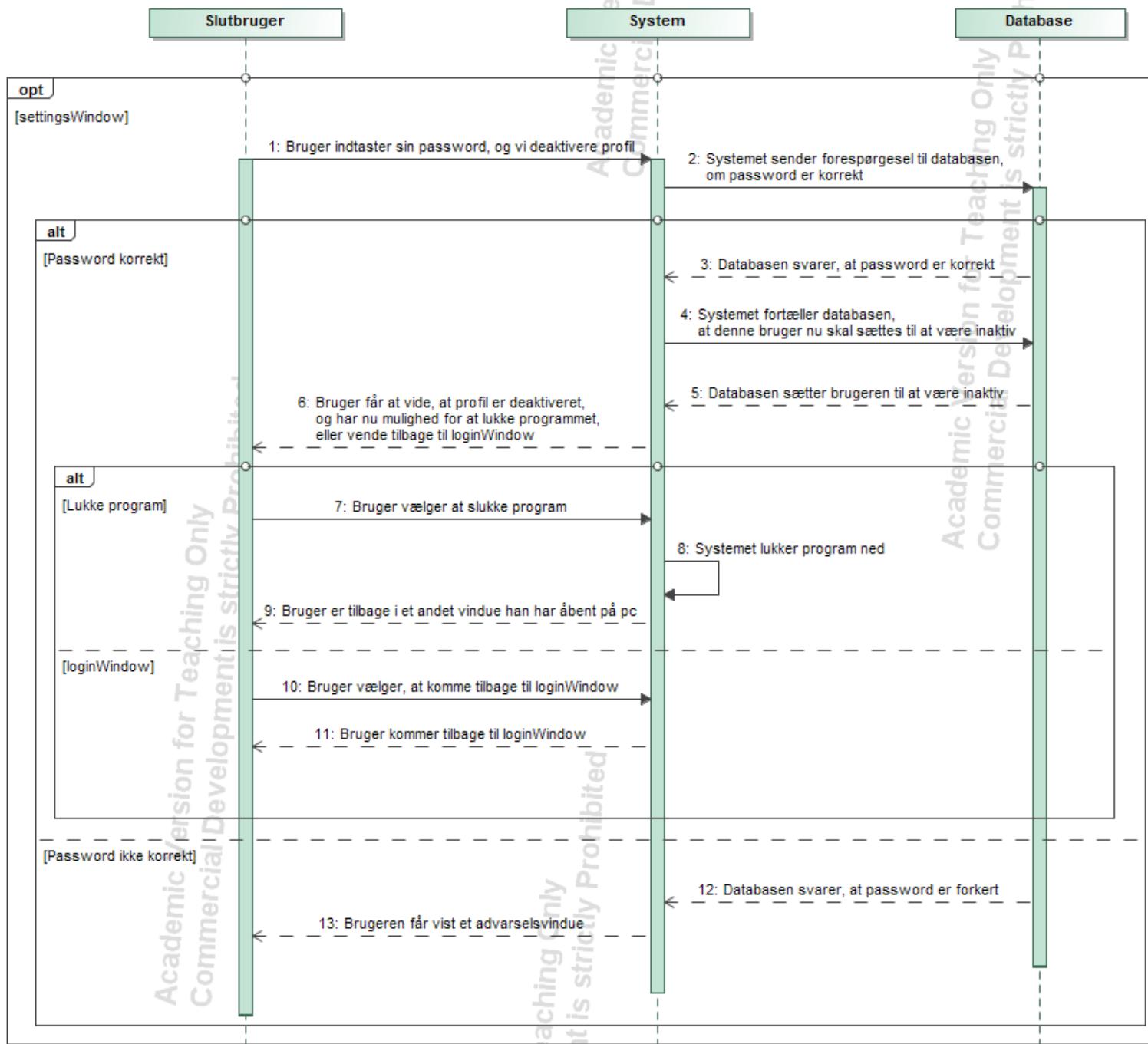


#### 5.4.6.

#### Deaktivér bruger

Bruger indtaster her sin password, for at kunne deaktivere sin profil. Systemet vil sende forespørgsel til databasen, om hvorvidt password er korrekt eller ej. Hvis password er korrekt, har brugeren mulighed for enten at lukke programmet helt ned, eller komme tilbage til vinduet, hvor man logger ind. Hvis password er forkert, vil brugeren få vist et pop-up vindue, som fortæller passwordet er forkert.

interaction Deaktiver bruger [  Deaktiver bruger ]



## 5.5. 3-lagsmodellen

Vi har overholdt 3-Lags modellen, i det at vi har opdelt vores koder i et interface-lag, funktions-lag og data-lag.

Vores interface-lag er alt i SceneBuild\_JavaFX packagen. Funktions-laget er alt i QuickConnect packagen, undtagen user-, main- og controller-klassen.

Data-laget er user-klassen og databasen.

Den primære del af vores funktions-lag er Funktion-, FunktionUser- og Conntector-klassen. Connector-klassen og user-klassen ligger i funktion-klassen som objekter

Connectoren og Funktion fungerer som en bro mellem funktions-laget og data-laget. Connectoren forbinder til database og user-klassen indeholder brugeroplysninger,

som i begge tilfælder er en del af data-laget. For at tilgå databasen eller user-klassen, skal man igennem Funktion-klassen først, og det gælder alle interface-klasserne også.

På den måde har vi overholdt 3-lags modellen. Følgende billede viser starten af vores JavaFX applikation og illustrere 3-lagsmodellen implementeret i vores program.

```

39  @Override
40  public void start(Stage stage) {
41      User user = new User();
42      Function f = new Function(user);
43      this.fu = new FunctionUser(f);

```

## 5.6. Nem-ID validering

For at få adgang til NemID's system, kræver det at man har et CVR nummer, og derfor har vi oprettet et CVR nummer: Quiconnect CVR-nr. 37588717 Derudover, kræves det at vi har adgang til PID/RID tjenester, for at få adgang til individuelle personer's oplysninger. For at få adgang til det skal vi have en virksomhedscertifikat og sende en anmodning om adgang til PID/RID tjenester, hvilket tager flere uger at gennemføre. For at få adgang til at anmode om PID/RID tjenester, skal man have NemID erhversnøglekort, hvilket vi modtog for nogle dage siden. NemID kan implementeres i den næste version af vores program, men i aktuelle version kan vi ikke nå at implementere det, da der er for meget ventetid fra digitaliseringsstyrelsen. Det kræver også en masse tid at lære NemID's API, efter vi har fået adgang til den.

# 6. Test

## 6.1. Test case 1: Ændre password test

Vi har oprettet test for vores changePassword funktion. Dette gør vi for at sikre at når funktionen bliver kørt korrekt så er det gjort korrekt.

Vi starter med at lave et objekter af user, function og functionuser klassen. Derefter definerer vi old og new password, for vores changePassword() funktion har parameterne oldPass og newPass. Vi sætter user\_id til objekt til at være 1, fordi det er den bruger vi gerne vil ændre passworden til.

Vi starter med printe den gamle kode i konsollen ud i MD5 – krypteret password. Derefter kører vi funktionen som skal ændre den gamle kode fra at være Test1234 til newTest1234. Så printer vi den nye krypteret kode fra databasen og sammenligner den med den f.md5(newPass)"newTest1234".

Hvis de sidste to sidste print er de samme betyder det at funktionen changePassword fungerer korrekt.

Console:

```
2c9341ca4cf3d87b9e4eb905d6a3ec45--> OLD PASSWORD MD5
65dc21cb4047b7e8292302df2c2e38d1 --> NEW PASSWORD IN DATABASE MD5
65dc21cb4047b7e8292302df2c2e38d1 --> NEW PASSWORD MD5
```

Testet metoder:

```
changePassword(String oldPass, String newPass, String newPass2){...};  
checkPassword(String pass){...};
```

## 6.2. Test case 2: Registrering test

Vi starter med at lave et objekter af user, function og functionuser klassen. Vi fjerner først brugeren med navnet "test", hvis brugernavnet allerede eksisterer i databasen.

Vi initialiserer variabler for at definere datoer, som skal sættes ind i databasen når brugeren bliver oprettet. Vi initialiserer int variabel i, og sætter den til at være checkRegister funktionen, da den returnerer en int, den kan returnere 10 forskellige tal, afhængig af hvilken betingelser man ikke har opfyldt under registreringen, men returnerer 10, hvis betingelserne er opfyldt.

Hvis checkFunction returnerer 10 så skal function addUser køres, som en ny bruger ind i users tabel i vores database.

Hvis testen køres uden fejl, vil den printe emailet af den bruger som der lige er blevet oprettet.

*Console:*

```
10
test123@gmail.com
```

*Testet metoder:*

```
checkRegister(String user, String pass1, String pass2, String email, LocalDate date){...};
addUser(String user, String pass, String email, int age){...};
```

### 6.3. Test Case 3: Venneanmodning test

Her tester vi, for at se om, 2 forskellige brugere kan blive venner med hinanden, acceptere venneanmodninger, og derefter se om venskabet bliver gemt i databasen. Den starter med at slette brugerens venskab i databasen, hvis det eksisterer i forvejen.

Derefter tilføjer test1, test2 som ven.

Så printer vi test1's venneliste, som til at starte med er 0, da test2 ikke har accepteret venneanmodningen.

Derefter skifter vi over til test2, som skal acceptere venneanmodningen af test1, også skifter vi over til test1 og printer test1's venner. Derefter kan vi se, at test1 har test2 som kontakperson.

Console:

```
ingen venner
test2 Vennerliste med accept
```

Testet metoder:

```
addFriend(String username){...};
getAllFriendsNickname(){...};
acceptFriend(String requestName){...};
```

## 6.4. Test Case 4: Besked test

En anden stor opgave i vores program var, at man skal kunne sende og modtage beskeder. Her har vi lavet en testcase, som viser beskederne mellem 2 brugere.

Vi starter først med at angive at den nuværende bruger er test2, og derefter sletter alle de beskeder, som test2 har sendt til test1.

Så sender vi 9 beskeder til test1, som også bliver vist i konsollen.

Herefter printer vi alle de beskeder der bliver sendt til bruger 1, samt det brugernavn som har sendt beskederne. Dette gør vi ved at lave 2 forloops, da getAllMessages returnerer en multidimensional ArrayList.

*Console:*

```
SENDT: (message,user_ID,message_sent,receiver_id) VALUES ('BeskedTest1 ','2','1466100827','1')
SENDT: (message,user_ID,message_sent,receiver_id) VALUES ('BeskedTest2 ','2','1466100827','1')
SENDT: (message,user_ID,message_sent,receiver_id) VALUES ('BeskedTest3 ','2','1466100827','1')
SENDT: (message,user_ID,message_sent,receiver_id) VALUES ('BeskedTest4 ','2','1466100827','1')
SENDT: (message,user_ID,message_sent,receiver_id) VALUES ('BeskedTest5 ','2','1466100827','1')
SENDT: (message,user_ID,message_sent,receiver_id) VALUES ('BeskedTest6 ','2','1466100827','1')
SENDT: (message,user_ID,message_sent,receiver_id) VALUES ('BeskedTest7 ','2','1466100827','1')
SENDT: (message,user_ID,message_sent,receiver_id) VALUES ('BeskedTest8 ','2','1466100827','1')
SENDT: (message,user_ID,message_sent,receiver_id) VALUES ('BeskedTest9 ','2','1466100827','1')
BeskedTest1 BeskedTest2 BeskedTest3 BeskedTest4 BeskedTest5 BeskedTest6 BeskedTest7 BeskedTest8 BeskedTest9
test2test2test2test2test2test2test2test2test2test2
```

*Testet metoder:*

```
sendMessage(String msg, int receive_id){...};  
getAllMessages(int id){...};
```

## 7. Konklusion

Programmet, som der er blevet bedt om, er lavet. En del både specifikke og uspecifikke krav, er blevet opfyldt, og ønsker er imødekommet. Projektlederen valgte, at vi skulle lave GUI'en ved hjælp af JavaFX. Vi har derved lavet nogen forskellige knapper, søgefelter osv. Med hensyn til GUI'en, har vi faktisk formået at overholde alt, som vi er blevet stillet af krav.

3-lagsmodellen er blevet overholdt med hensyn til koden. Vi har nemlig netop et Interface-lag, Funktions-lag og Data-lag. Som det er blevet beskrevet tidligere i rapporten, har vi SceneBuild\_JavaFX pakken, som Interface-lag, QuickConnect pakken, som Funktions-lag, og til sidst user-klassen og databasen, som vores Data-lag. Grunden til de forskellige dele, er netop beskrevet tidligere.

Ved hjælp af vores analyse-del og design-del har vi formået, at blive færdig med programmet, så det faktisk fungerer, som vi ønskede. Vores database er blevet korrekt sat op, og koden er, som den skal være. Hvis man ser vores kravspecifikation igennem, kan man bekræfte, at de forskellige krav er blevet opfyldt. Dog har vi ikke Google Web Tool inddraget i vores program, hvorfor vi har valgt at lave en e-mail validering, og en Facebook validering. Ved hjælp af disse, kan vi verificere, at personerne, som prøver at registrere sig, er troværdige personer, og ikke bare personer, som opretter profiler "for sjov".

## 8. Konfiguration

### Udviklingsplatforme

Under udviklingen af programmet, har vi benyttet os af forskellige software platforme.

Til analyse og design af projektet har vi benyttet os af programmet Magic Draw (version 18.1).

Til udarbejdelsen af rapporten har vi brugt Microsoft Office Word (Version 15.11.2 (150701)).

Til udvikling af programmet har vi brugt programmeringssproget Java. ECLIPSE IDE (Integrated Development Environment) Version: Mars. 2 Release (4.5.2) er blevet brugt som udviklingsværktøj.

Vi har brugt JavaFX, version 8.0.66-b33, til at udvikle vores GUI.

Til at håndtere og tjekke databasen har vi brugt MySQL Workbench Community (GPL) for version 6.3.6.

Vi har også en database liggende på Gigahost.dk, som indeholder data, der kræves for at køre/bruge programmet. Databasen er MySQL(5.6.25) server, hvor man kan bruge sql statements til at håndtere data, som programmet skal bruge.

Vi har dermed brugt en MySQL connector til Java(mysql-connector-java-5.1.26-bin.jar), til at forbinde til vores database via vores program. Derudover er der også blevet brugt en Mail Connector til Java(mail.jar), for at kunne sende mails under registreringen. Den sidste connector fil som er brugt er programmet er en Facebook connector(restfb-1.24.0.jar), som bruges til at registreringen med Facebook.

Connector filerne ligger er vedlagt i zip filen. For at disse connectere skal køre skal de ligge i rodmappen. Dermed skal de importeres i Eclipse. Dette kan gøres ved: Højre klik på JRE System Library > Build Path > Configure Build Path > Add JARs... > 17CDIO\_Final, og dermed vælge de jar filer, som du vil importere.

De operativsystemer der blevet brugt til udvikling er Windows 10 og Mac OS X El Capitan (v. 10.11.1). For at køre programmet skal man have en maskine der kan køre Java applikationer(Windows, Mac, Linux). Derudover skal der på operativ systemet skal der som minimum installeres Java SE version 1.8. Dermed skal man også have mulighed for at køre den færdige .jar fil i konsollen/terminalen/Command Promt. Java-projektet er komprimeret som ZIP-fil. Det kan importeres som et færdigt projekt i Eclipse ved at udpakke ZIP-filen. Man skal blot gøre følgende: Import > General > Existing Projects into Workspace > Browse > Vælge den udpakkede ZIP-fil > Finish.

For at køre QuickConnect, skal man vælge Java perspective, hvor man derfra kan se en Package Explorer. Her kan man finde det importerede projekt. Man går ind i følgende: 17CDIO\_Final > src > QuickConnect > Main.java. Ved at trykke på højre-klik på Main-klassen trykker man så på Run as > Java Application, hvilket resulterer i, at programmet åbnes.

## 9. Referencer

### 9.1. Hjemmesider

- 9.1.1. [www.stackoverflow.](http://www.stackoverflow.com)

### 9.2. Programmer

- 9.2.1. Magic Draw (version 18.1)
- 9.2.2. Microsoft Office Word (version 15.11.2 (150701))
- 9.2.3. Eclipse IDE (version: Mars. 2 Release (4.5.2))
- 9.2.4. MySQL Workbench Community (GPL) (version 6.3.6)