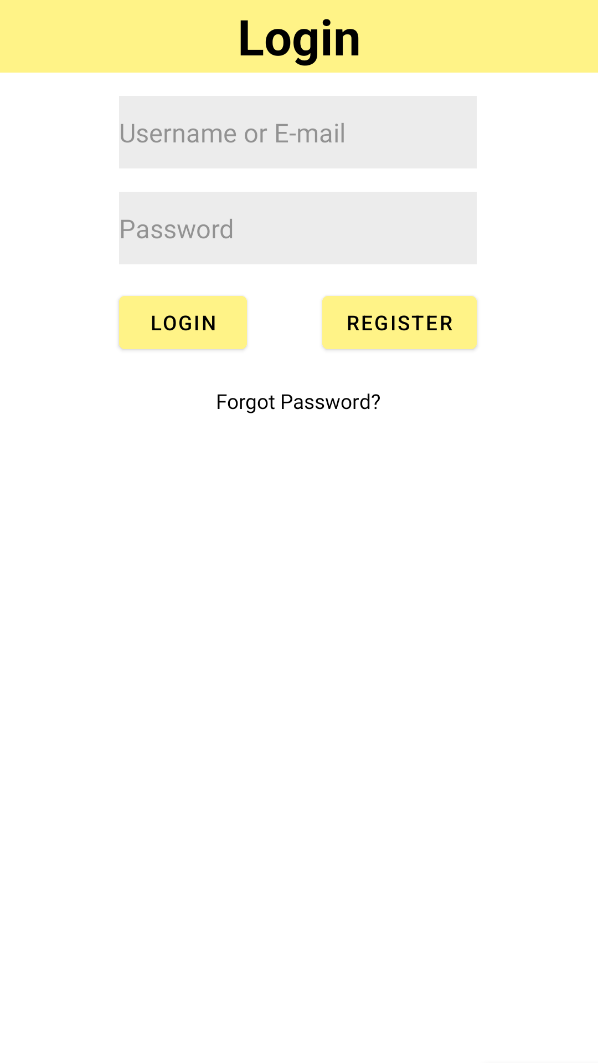
El-installatør app rapport

Førsteårsprøve af Tim, Simon og Kristian

Antal tegn inklusive mellemrum: 63.491

Github repository: <https://github.com/Kristian-j-k/Elinstallator_hold4>



**Indholdsfortegnelse**

[Indledning 3](#_Toc105576240)

[Problemformulering (Simon) 3](#_Toc105576241)

[Virksomhedsbeskrivelse (Simon) 4](#_Toc105576242)

[GDPR (Kristian) 8](#_Toc105576243)

[Metodevalg (Simon) 8](#_Toc105576244)

[Teknologi 11](#_Toc105576245)

[Database E/R Diagram (Kristian) 11](#_Toc105576246)

[Operativsystemer (Simon) 18](#_Toc105576247)

[Programmering 19](#_Toc105576248)

[Email (Simon) 19](#_Toc105576249)

[Design mønster (Kristian) 21](#_Toc105576250)

[Programmering af variabler (Kristian) 21](#_Toc105576251)

[Database (Tim) 24](#_Toc105576252)

[Login (Tim) 26](#_Toc105576253)

[Registration (Tim) 28](#_Toc105576254)

[Generering af pdf-fil (Kristian) 29](#_Toc105576255)

[Systemudvikling 29](#_Toc105576256)

[Kravindsamling og -specifikation 29](#_Toc105576257)

[Use cases (Simon) 29](#_Toc105576258)

[Object-Oriented Analysis (OOA) 30](#_Toc105576259)

[System sekvens diagrammer (SSD) (Simon): 30](#_Toc105576260)

[Domænemodel (Simon) 36](#_Toc105576261)

[Object-Oriented Design (OOD) 37](#_Toc105576262)

[Klassediagram (Tim) 37](#_Toc105576263)

[Sekvensdiagram (Kristian) 38](#_Toc105576264)

[Kvalitetssikring 39](#_Toc105576265)

[Brugertest med projektstiller (Kristian) 39](#_Toc105576266)

[Brugergrænseflade test (Kristian) 40](#_Toc105576267)

[Konklusion (Simon) 41](#_Toc105576268)

[Bilag 42](#_Toc105576269)

# Indledning

Mange virksomheder kan med fordel få udviklet sig en app til at hjælpe dem i deres daglige arbejde, eller få deres kunder tættere på deres produkt. I dette projekt undersøger vi hvordan vi kan støtte el-installatørerne på Zealand bedst, med en app der gør deres dagligdag nemmere.

### Problemformulering (Simon)

El-installatørerne på Zealand oplever at de spilder for meget tid på at skrive ting ind i deres skema, som stadig er på papir-form. Når skemaet er i papir-form, kan det også blive væk eller ødelagt. Disse problemer vil vi undersøge hvordan vi kan løse med en app, der kan generere et skema som el-installatørerne derefter kan udfylde, få lavet en .pdf-fil af skemaet som bliver gemt på telefonen, og til sidst sende dette digitale skema videre til en kunde eller andet over e-mail. Vi vil gøre dette ved at tage brug af de værktøjer vi har lært på 1. og 2. semester af vores uddannelse. Dette gør det også muligt for os at opfylde læringsmålene som de står beskrevet i studieordningen.

### Virksomhedsbeskrivelse (Simon)

Vores projekt betragter vi som en god mulighed for at danne vores egen virksomhed, hvis arbejde det ville være at udvikle apps til android og Apple telefoner. Vi vil efter projektet viderebygge på vores app til Zealand for, så vi kan lave en generisk der let kan sælges til andre virksomheder.

Vores firma vil blive skabt som et Interessant Selskab (I/S), da et Interessant Selskab gør det muligt at skabe firma som to eller flere personer, uden man har brug for en startkapital. I stedet vil vi som ejere skulle hæfte personligt og solidarisk. Der er ulemper med at skulle hæfte for alt personligt, men vi mener stadig at dette er den bedste mulighed for os, på nuværende tidspunkt. På sigt kan vi eventuelt konvertere virksomheden til Anparts Selskab (ApS), der vil gøre os i stand til at hyre flere medarbejdere og vi vil slippe for det personlige ansvar.

**Mission**

Vores mission er at skabe god software som vil kunne bruges af mange forskellige mennesker rundt omkring i verdenen.

**Vision**

Efter vores gratis udvikling af vores til Zealand, vil vi undersøge om der er andre virksomheder rundt omkring i Danmark, der kunne ønske sig en App der minder om den vi har lavet. Hvis disse virksomheder er interesserede, ville vi nemt kunne lave en modificeret version af appen og sælge den til disse virksomheder.

**Forretningsmodel**

Vores produkt er en Android app, som vil kunne benyttes af en El-installatør til at skabe et skema/checkliste, som el-installatøren vil kunne bruge når personen er ude for at inspicere forskellige slags elinstallationer.

Vi kender kender til en virksomhed, kaldet Inspectly, som har lavet en App der minder om den vi er i gang med at lave til el-installatørerne på Zealand. Vi kender ikke til andre virksomheder end Inspectly, så derfor vil vi gå ud fra at de fleste el-installatører enten bruger Inspectly eller et skema på papirform i deres dagligdag.

Vi mener derfor at der er en oplagt forretningsidé i, at vi samler erfaring ved at bygge vores App til Zealand, samtidig med at vi tænker på hvordan vi nemt senere kan ændre på Appen, så den ville appellere til andre virksomheder inde for El-installatør feltet, med den udfordring i skema i papirform. Det er derfor vigtigt at selvom vi giver vores App gratis til Zealand, at vi stadig beholder retten til at videreudvikle på voress App, så vi senere at videresælge en modificeret app til andre virksomheder.

Vi har i forbindelse med udviklingen af vores forretningsmodel, benyttet et business model canvas (BMC) til at hjælpe os med at få overblik over de forskelige nøgleområder i vores virksomhed.

Et business model canvas (BMC) består af følgende elementer:

**Partnere (Partners)**

I partnerdelen forklarer vi hvilke forskellige partnere arbejder vi sammen med i vores virksomhed. Dette kan være i forbindelse med for eksempel markedsføring, produktion eller andet. Vi har i vores tilfælde kun en partner, hvilket er el-installatørerne fra Zealand, som også er vores kunde. Vi betragter dem som partner selvom de også er kunde, fordi de bidrager med viden omkring deres branche som er nødvendig for at vi kan udvikle en vel fungerende app.

**Aktiviteter (Key Activities)**

Aktivitet delen handler om hvilke kerneopgaver vi som virksomhed kan løse. Vores virksomhed er specialiseret i at udvikle apps til Android, ved brug af Java. Og i dette tilfælde udvikler vi en app, der er rettet imod virksomheder som ansætter el-installatører.

**Ressourcer (Key Resources)**

Key ressources handler om hvilke ressourcer vores virksomhed har til rådighed. For eksempel kan dette være økonomiske ressourcer, fysiske eller hvor mange ansatte vi har. I vores virksomhed har vi 3 mand til rådighed, og vi deler arbejdet så lige som det er muligt. Økonomisk vis, så arbejder vi hjemmefra og gratis. Vores fysiske ressourcer er vores allerede betalt for hjemme computere vi arbejder fra. Vi benytter os også af gratis software som Android Studio til udviklingen af vores app.

**Kundeværdi (Value Propositions)**

Kundeværdi handler om den ydelse vi sælger, og hvilken værdi det kan give vores kunder. Vi sælger vores viden i programmering, softwareudvikling og tid, og dermed kan vi løse vores kunders behov. I dette tilfælde ligger værdien i at vi kan erstatte el-installatørens papir-form skema med vores android app.

**Kunderelation (Customer Relationships)**

Kunderelation handler om hvordan vi kan stå til rådighed for vores kunder. Vi vil sørge for at holde en tæt kundekontakt med ugelige møder, og hvis det er nødvendigt med kontakt over mail under appens udviklingsproces. Derudover kan vi også stå til rådighed efter vi har overleveret appen, for at enten rette fejl i appen, eller måske tilføje nye egenskaber og funktionaliteter.

**Kanaler (Channels)**

(Handels)kanaler beskriver hvordan vi forhandler med kunderne. Vi har tænkt os at kontakte forskellige firmaer med el-installatører ansat, og demonstrere vores idé og app, enten fysisk eller eventuelt et online møde. At opsøge potentielle kunder er en nem måde at skaffe kunder på, når ens firma og produkt ikke har noget omdømme i forvejen.

**Kunder (Customer)**

Vores nuværende kunde er Zealand, hvor vi specifikt arbejder med el-installatørerne på Zealand. Fremtidige kunder vil være firmaer der har ansat el-installatører.

**Omkostninger (Cost structure)**

Her identificere vi hvilke omkostninger vi vil få. På nuværende tidspunkt har vi ingen omkostninger andet end vores tid. Men på sigt vil der eventuelt opstå omkostninger ved leje eller køb af kontorlokaler hvis vi beslutter os for, at vi ikke vil arbejde hjemmefra. Der vil også være omkostninger i form af løn, køb af software til udviklingen af vores produkter, forskelligt materiale til kontoret og computer hardware.

**Indtægter (Revenue streams)**

Her identificerer vi hvordan vi har tænkt os at tjene på vores produkt. Appen vi laver til el-installatørerne på Zealand laver vi gratis, men på sigt vil vi gerne bruge appen vi udvikler til Zealand som et standard produkt, vi kan ændre på og tilpasse så vi appen kan sælges til andre virksomheder der ansætter el-installatører. Dette salg kunne eventuelt laves som et direkte køb, hvor at kunden overtager appens drift og alt andet, eller kunden kunne hyre os til at tilpasse appen og holde den kørende.

**Finansiering (Financing)**

For at få styr på vores finansiering, har vi lavet et investeringsbudget. Men dette er vores overordnede tanker.

Indtægter: Vi vil lave et software program, som vi vil kunne tilpasse til den enkelte virksomhed som ansætter el-installatører. For køb af denne app har vi sat en pris på 30.000 DKK for et køb af licens og tilpasningen af appen til den enkelte kunde. Og derudover vil vi også tilbyde kunder at købe en support/vedligeholdelses aftale, hvor vi vil tilbyde at opdatere appen som kunden har lyst til, eller hjælpe kunden hvis der går noget galt med appen. Denne support aftale vil koste kunden 5000 DKK pr. halvår. I det første halvår udover vores semester projekt, bruger vi tiden på at lave vores app om så den hurtigt kan tilpasses en ny kunde, og i det andet halvår forventer vi at have fået vores første kunde.

Udgifter: Vi vil kunne holde vores indledende omkostninger og udgifter lave da vi stadig er i gang på Datamatiker studiet, og denne virksomhed vil være et projekt ved siden af studiet. Vi har ingen udgifter i form af løn eller husleje, og derfor vil også vores omkostninger i form af bogholderi og diverse være lave. Siden at vi forventer et salg efter første halvår, forventer vi at kunne give os selv løn, men vi vil investere det meste af vores overskud i virksomheden.

Opstartslån: Vi mener ikke vi har brug for at skulle låne penge nogen steder for at få startet vores virksomhed, da vi allerede har det hardware og software i form af computere og IDEs, vi skal bruge for komme i gang med at udvikle vores produkt.

**Strategi for implementering og skalering af virksomheden til andre kundesegmenter**

Implementering: Vores implementering af appen forventer vi at gå nemt, eftersom at vi vil være i kontakt med vores samarbejdspartner, el-installatørerne på Zealand, under hele projektets udviklingsproces. Vi vil overlevere appen og vil hvis nødvendigt oplære el-installatørerne i hvordan man bruger appen korrekt.

Skalering: Vi har brugt noget tid på at undersøge om vores virksomhed har nogen konkurrenter, og har fundet mindst en virksomhed som vil være en konkurrent. Virksomheden kaldes for Inspectly og tilbyder en app med omtrent de samme features som vores egen app, med mere. Derfor vil vores strategi for skalering være at lave den perfekte app, så virksomhederne vil vælge os frem for Inspectly, men også at tage kontakt til nyligt oprettede virksomheder, som endnu ikke har besluttet sig om de skulle f.eks. vælge at købe sig ind i Inspectlys app.

### GDPR (Kristian)

For at overholde GDPR overvejer vi om vi har personoplysninger og om vi har følsomme personoplysninger. Ifølge tjekskemaet indsamler vi oplysninger som kundenavn og adresse. Er kunden en person er det en ikke følsom personoplysning. Vi skal også have identifikation af installatør og hvem den er udført af. Igen er det personoplysninger. I programmet har vi login med mail og adgangskode, så vi beskytter persondataene. Da vi bruger e-mail til login, skal disse e-mailadresser også beskyttes.

Vi skal begrænse indsamlingen af data om brugere og kunder, og må ikke viderebehandle data til andre formål end aftalt med kunder og brugere. Derfor kan vi ved første login på programmet bede om godkendelse af at Zealand må sende reklame på den e-mail som er tilknyttet brugere. Får Zealand ikke denne tilladelse og alligevel benytter brugernes e-mail til reklame, risikere Zealand at være skyldige i at bryde GDPR ved at benytte data, nemlig e-mailadresserne til andet end det nødvendige og til andet end hvad de har informeret om. I den mere grelle overtrædelse af GDPR vil personer med adgang til programmets database kunne sælge e-mails eller informationerne fra tjekskemaerne til tredjeparter.

Det vurderes at det først og fremmest er vigtigt at installatørerne som benytter programmet generelt, overholder GDPR i deres kundeaftaler og at programmet i de fleste tilfælde vil komme under deres generelle aftale så længe at programmet fungerer med opbevaring af data hos de respektive installatører. Ændres programmet til at gemme data centralt hos Zealand vil det kræve information om deling af data med tredje part.

### Metodevalg (Simon)

Under vores projekt, har vi taget brug af den iterative udviklingsproces, arbejdet agilt på projektet og har taget brug af SCRUM.

At arbejde med den iterative udviklingsproces betyder, at vi arbejder med ugelige iterationer af vores program og artefakter, så vi en dag hver uge udvælger de dele vi mener skal bruges i den næste iteration af programmet. Denne dag bruger vi også på at finde ud af hvad vi skal bruge den næste uge på.

Vi valgte også at afholde et møde hver anden dag på discord, hvor vi har snakket om hvordan det har gået med vores igangværende opgaver, og vi har haft problemer osv.

Når en af os har lavet et artefakt, stykke dokumentation eller noget kode, så sørger vi for at det sættes i en pull request i vores repository på Github så vi kan review hinandens arbejde. På denne måde kan vi bedre sikre kvaliteten af vores arbejde og minimerer de fejl vi ellers ville begå.

**Projektstyring**

* **Planlægning**

Vi har brugt Github project til at hjælpe os med at planlægge og styre vores projekt. Her laver vi issues som man derefter kan sætte sig selv på.

At bruge GitHub projekt er en nem og gratis måde at projektstyre på, og det har værktøjer vi kan bruge til at lave pull requests og review vores artefakter og kode.

Vores projekt er opdelt i forskellige faser:

1. To do: Issues der ikke er blevet tildelt, ligger under To do.

2.In progress: Hvis en issue er blevet tildelt en udvikler, vil denne issue blive flyttet til In progress

3. Review: Når et artefakt eller et stykke kode er færdigt, vil der blive lavet en pull request, og denne pull request vil blive reviewed af de andre i projektet. Hvis der ikke er nogen mangler, så vil pull request blive accepteret og det vil blive merged ind i main branch.

4. Done: Alle issues der har haft deres pull request accepteret, vil automatik blive sat ind under Done.

Et billede, der indeholder tekst, elektronik

Automatisk genereret beskrivelse[[1]](#footnote-1)

* **Risikoanalyse**

Vi har som en del af styringen af vores projekt, lavet en risikoanalyse som går over de risici der kommer ved at arbejde på et projekt som dette. En risikoanalyse giver os en ide om hvad vi kan forvente af problemer på projektet undervejs, og giver os muligheden for at tage højde for disse potentielle problemer når vi planlægger vores projekt. Et eksempel kan fx være sygdom under projektet, der kan være med til at forsinke vores arbejde.

Et billede, der indeholder tekst, skærmbillede, monitor, skærm

Automatisk genereret beskrivelse[[2]](#footnote-2)

**Github**

På Github ligger vi vores kode artefakter. Vi har delt vores repository op i flere forskellige mapper for at et bedre overblik over projektet.

Vi har 01 Business model mappen hvor alt der har ved virksomheden ligges ind.

Vi har 02 Requirements mappen hvor ting som kravspecifikation, use cases og risiko analysen bliver lagt ind.

I 03 OOA mappen ligger vi alle vores analyse artefakter.

I 04 OOD mappen ligger vi alle design artefakter.

I Rapport mappen ligger vi vores uddeling af emner vi skal skrive om, og vores færdige rapport.

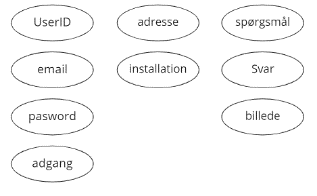
Vores kode skulle helst ligge i en mappe kaldet 05 Implementation.

# Teknologi

### Database E/R Diagram (Kristian)

Et billede, der indeholder tekst, whiteboard

Automatisk genereret beskrivelseFor at designe databasen har vi analyseret behovet for attributter ud fra domænemodellen. Vores første domænemodel fokuserede på bruger, med mange forskellige slags brugere. Men vi fandt hurtigt ud af at alle typerne kan gå under én slags bruger med forskellige rettigheder.

Desuden kom der krav til tjeklisterne om blandt andet at indeholde billeder. Derfor har vi fået attributterne: brugerID email pasword til bruger adgang, og adgang som adgangskontrol. På tjeklisten som i vi i starten kaldte for assignment er der flere attributter, men nogle vigtige er adresse og installation, spørgsmål og svar og billeder.

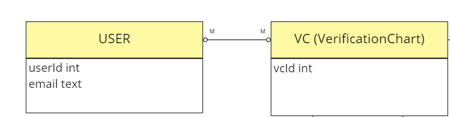
**E/R diagram**

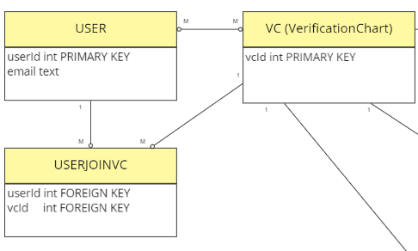
Efter at vi har fået oversigt over attributterne, også kaldet egenskaber, kan vi udarbejde et E/R diagram. Diagrammet giver en god oversigt over relationerne mellem de forskellige enheder i databasen. Vi har valgt at benytte diagrammet for at få placeret de forskellige enheder på en god måde i forhold til hinanden.

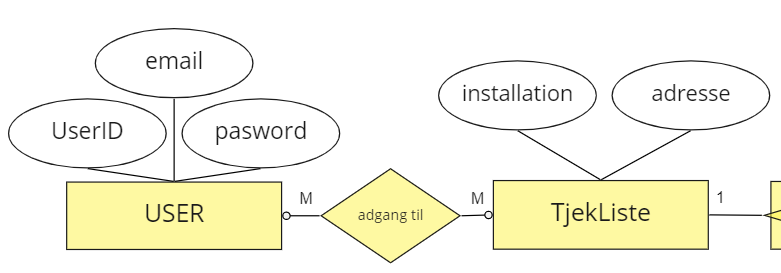
Egentligt er programmet ikke afhængigt af at benytte en database, da det er muligt at udfylde tjekskemaet og med det samme at omdanne det til en pdf og at sende det på mail. Men fordelen ved databasen er at den kan langtidsopbevare dataene, som ellers vil gå tabt når programmet lukkes. Ved at langtidsopbevare dataene bliver det muligt for brugeren at stoppe udfyldningen af skemaet, lukke programmet, og at åbne programmet igen på et andet tidspunkt og fortsætte med at udfylde skemaet.

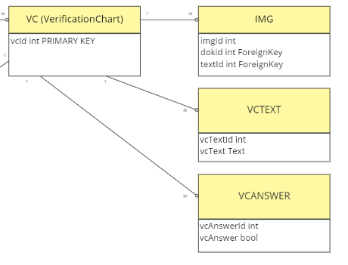
Databasens formål centrerer sig om at kunne gemme indtastningerne til tjekskemaet. Der er dog en fordel mere ved at benytte databasen, vi kan nemlig centralisere dataene. Det betyder at vi i stedet for at gemme indtastningerne fra tjekskemaerne lokalt på den enhed som indtastningen er sket på, kan have dataene gemt i skyen, og lade brugeren tilgå dataene i skyen via sin enhed. Formålet med denne placering af databasen er at der nu kan være flere brugere som har adgang. Det betyder også at der ikke vil være brug for afsendelse af e-mails mellem medarbejdere, da alle medarbejdere vil kunne se tjekskemaerne.

Den centrale placering af databasen giver dog en sikkerhedsudfordring og må derfor beskyttes med adgangskode. Håndteringen af adgangskoder og brugere skal databasen også forberedes til.

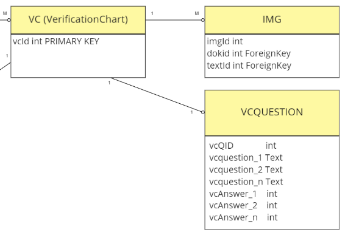
Nedenfor ser du to vigtige entiteter, eller enheder i databasen, tabellen for brugere og tabellen for tjekskemaerne. Vi har gjort os overvejelser om hvor mange brugere der er pr tjekskema, og hvor mange tjekskemaer der er pr bruger. Konklusionen var at vi gerne vil gøre det muligt for brugeren at have flere tjekskemaer at arbejde på samtidig, uden at ét tjekskema skal færdigmeldes før et nyt kan åbnes. Derfor er tabellen for tjekskemaer (VC) markeret med et M for mange. En bruger kan altså have adgang til mange tjekskemaer.

Bruger tabellen (USER) er også markeret med M for mange. Det betyder at flere brugere kan have adgang til det samme tjekskema. Det betyder ikke at alle brugere får adgang til alle tjekskemaer, men databasen rummer muligheden for det. Derfor vil det næste være at lave en begrænsning på hvem som har adgang til hvilke tjekskemaer. Fordi brugerne har adgang til et udefineret antal tjekskemaer, og hvert tjekskema kan have et udefineret antal brugere, skal tabellen for adgangsbegrænsning tage højde for det og må nødvendigvis være variabel. Den lægges ikke sammen med hverken bruger eller tjekliste, men oprettes som en selvstændig tabel som sammenkæder bruger med tjekliste. Bruger tabellen (USER) er oprettet med et id som refererer til den specifikke bruger, dette id markeres med Primær nøgle (PRIMARY KEY) fordi tabellen kan stå alene, det samme gælder for id’et for tjeklister. I join tabellen som sammenkæder brugere med tjeklister sættes to værdier, id’et som refererer til en bruger, og id’et som refererer til en tjekliste. Fordi begge id først er defineret i andre tabeller, markeres de i join tabellen som fremmed nøgler (FOREIGNKEY). Det medfører at join tabellen nu er afhængig af de andre to tabeller, den har ikke nogen funktion hvis den stå alene. Det betyder at den kun bruges hvis der er oprettet en bruger den kan få et bruger id fra, og en tjekliste som den kan få et tjekliste id fra. Slettes en bruger eller en tjekliste som join får en fremmed nøgle fra, skal rækken i join tabellen altså også slettes. Man kunne måske ønske at gemme historik over hvilke brugere som har haft adgang til hvilke tjekskemaer på hvilke tidspunkter, men det rummer join tabellen ikke information om når en bruger eller tjekliste er slettet. Vil man gemme en historik, bliver det i det her tilfælde i en helt anden tabel, det har vi dog ikke planer om i øjeblikket. Eftersom at tabellen UserJoinVc er relationen mellem de to tabeller tegnes den som en rombe der står på en spids. Den er nu selve relationen mellem de to og ikke en selvstændig tabel. Det fremgår af det nye diagram som har fået lidt andre navne for nemheds skyld.



Vi har overvejet om teksterne og svarene til tjeklisterne skulle have en kolonne per spørgsmål eller om det skulle være en række pr spørgsmål som så skulle kæde spørgsmål og svar sammen. Billedet til højre, er i den første del af planlægningen.

Men eftersom hvert svar har en direkte relation til hvert spørgsmål, var det naturligt at lade dem relatere til hinanden. Der kan måske være et spørgsmål uden et svar, men det giver ikke mening at have et svar uden et spørgsmål. Derfor fik spørgsmåls tabellen i starten lov at beholde sin relation til tjekliste, mens svar tabellen relaterede til spørgsmåls tabellen.

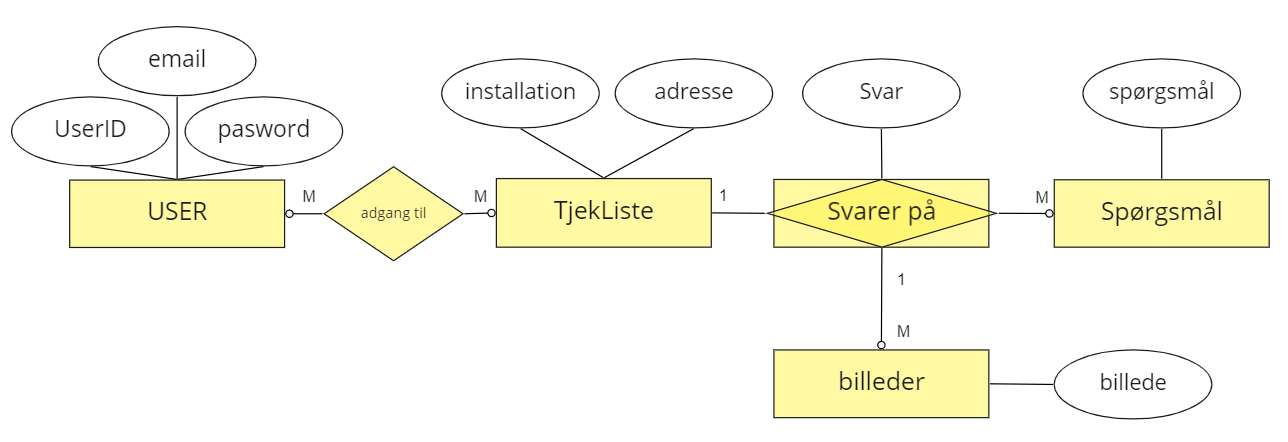
Spørgsmålene er foruddefinerede og for hvert spørgsmål er der et svar, svaret kan være ikke udfyldt, ja nej eller irrelevant, men der er kun et svar. Derfor ændres måden svaret lagres på til tal, så der kan lagres 0 1 2 eller 3 for ikke udfyldt, ja nej eller irrelevant. Relationen blev en én til én relation, det betød at der ikke skulle være to tabeller men kun en tabel, som indeholder alle data for de to tidligere tabeller.

Spørgsmålstabellen kom til at indeholde alle spørgsmål og alle svar. Den reserverede altså meget plads i databasen. Der er bare et problem, for spørgsmålene er foruddefinerede, og behøver ikke at blive gentaget. Man kunne argumentere for at der skal være rum for at gemme historikken med gamle tekster hvis teksterne bliver ændret, men det sørges der for når der afslutningsvist gemmes i pdf, og vi undgår redundans i databasen.

Der er undervejs kommet endnu en oplysning fra projektstiller, nemlig at der skal kunne tilføjes spørgsmål. Det kræver at databasen bliver mere dynamisk og tillader ikke at lægge svarene ind under tjeklistetabellen som ellers var oplagt når det så ud til at tjekliste og svar ville få én til én relation.

Designet af spørgsmål svar og Billeder er nu blevet revurderet på baggrund af behovet for den mere generiske tilgang.

Det har været svært ikke at lave et kompromis når spørgsmål svar og billeder har skullet tilføjes databasen.



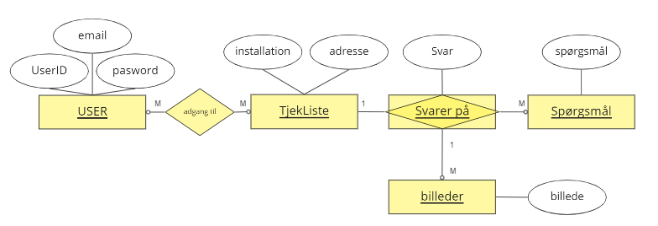
Spørgsmålene er tænkt som generiske sådan at der kan tilføjes og slettes spørgsmål efter behov. Spørgsmålene kan genbruge på flere tjeklister, og derfor er relationen mellem tjekliste og spørgsmål en mange til mange relation. Du har måske allerede bemærket at det ikke er det som står i diagrammet, her er den sat som en én til mange relation, og det er der en grund til.

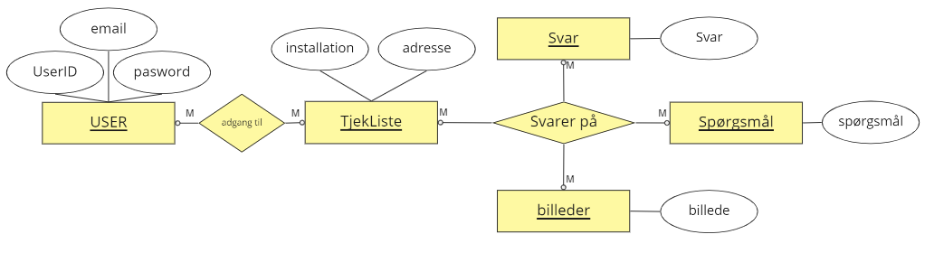
Det er et kompromis, og det er som om at uanset hvordan man vender og drejer det så resulterer det i et kompromis. Der er altså også andre muligheder man kan benytte. Udfordringen ligger i at selvom spørgsmålene kan bruges i mange tjeklister, så vil svaret kun høre til i én tjekliste, og det samme gælder billederne, med endnu en undtagelse, nemlig at de kun hører til i ét bestemt svar i én bestemt tjekliste. Valget er faldet på en udvidet relations tabel. Udover at relations tabellen kobler spørgsmål og tjeklister sammen, så indeholder tabellen også information om svar på spørgsmålet. Der er kun et svar per spørgsmål per tjekliste og nu er svar relationen blevet en én til mange, selvom den som relations kobling egentligt var en mange til mange. Den holdes unik sådan at spørgsmål ikke gentages på samme liste. Man kunne overveje om den i virkeligheden er en selvstændig tabel med en primærnøgle, her er problemet bare at et svar ikke kan eksistere uden et spørgsmål, den er altså afhængig af spørgsmålstabelen. En måde den kunne blive gjort selvstændig på vil være at lade den have sine egne spørgsmål som blev kopieret fra de prædefinerede spørgsmål. Men igen vil det være et kompromis at lave gentagelser af spørgsmålsteksten. Valget på at lave en kombineret relations tabel med ekstra tabeller er valgt for at gøre databasen så enkel som mulig. At svarene ikke ligger for sig selv ved siden af relations tabellen er fordi det ville være en én til én relation, hvilket vi normalt logger sammen som én tabel.

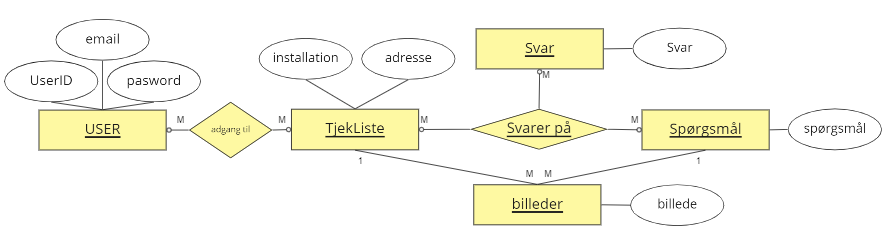
Det er muligt at lade billederne have relation til både tjekliste og spørgsmål med to fremmednøgler, men for enkelthedens skyld er de her relateret til svartabellen, det gør vi ved at give svartabellen et unikt id som billeder bruger som fremmednøgle. Nu kan vi tilføje et ubegrænset antal billeder til hvert spørgsmål, og vi kan holde styr på hvor hvert billede hører til, så de ikke bliver blandet sammen med billederne fra andre spørgsmål. I praksis vil vi i databasen blot notere stien og navnet på billedet som så gemmes separat i en mappe.

**Normalisering**

Generelt er der anvendt normalform 1 ved at tabellerne har nøglefelter. Vi har også gjort meget ud af at leve op til normalisering 1 ved at undgå gentagelser. For eksempel var det meget oplagt at have gentagelser af spørgsmålene for hvert tjekskema. Det har vi undgået ved at lave en tabel med spørgsmål og en anden tabel som linker tjekskemaerne med spørgsmålene efter behov. Havde vi lagt de 39 spørgsmål som sikkerhedsstyrelsen har defineret ind i tabellen for tjekskemaet, så ville vi ud over at det ville være voldsomt uoverskueligt også have mange spørgsmål som ikke er relevante eller som aldrig bliver besvaret. Samtidig ville det ikke være muligt at tilføje spørgsmål. Nu kan tjekskemaet variere i antal af spørgsmål uden at hver post varierer.

De fleste tabeller opfylder normalform 2 ved at have unikke nøgler. User har et unikt id for hver bruger og Tjekliste har et unikt id for hver tjekliste som genereres. Det samme gælder hvert spørgsmål som oprettes. Lidt anderledes er det for svar tabellen som fungerer som relation mellem tjekliste og spørgsmål og derfor har to fremmed nøgler, de to fremmednøgler kan tilsammen godt udgøre et unikt id, men der kan være en risiko for gentagelser. Gentagelser kan selvfølgelig undgås, men ved at lave et unikt id ved siden af fremmednøglerne, og for nemmere at kunne referere billeder til tabellen vil vi faktisk oprette den med et unikt id så også den lever op til normalform 2. For tabellen for billeder er det vigtigt at den har en fremmednøgle som refererer til et bestemt svar i en bestemt tjekliste, den kan klare sig uden et unikt id, men at tilføje et unikt id vil kun være en fordel da man dermed får mulighed for at lægge billederne i den rækkefølge man ønsker. Havde den ikke et unikt id men kun en gentaget fremmednøgle, ville man stadig kunne finde billederne frem, men rækkefølgen ville være tilfældig. På diagrammet er entiteterne som har primærnøgler nu understregning.

For at leve op til normalform 3 vil vi gerne overveje om der er tabeller som kan deles op i mindre tabeller. Fokus rettes mod relationen svarer på som også er en enhed med svar, disse svar er gentagelser af blandt andet ja og nej, og de lever heller ikke helt op til normalform 1 om at undgå redundans. Vi flytter svar mulighederne ud som en selvstændig enhed, og lader svare på være en ren relation, til fire forskellige enheder.

Vi kan nu svare både ja og nej på et spørgsmål, så vi har inkonsistens. Det løses ved at give relations tabellen en primærnøgle som består at tjekliste spørgsmål og svar, så der kun kan laves en kombination af hver. Nu kan vi kun tilføje et billede til hvert spørgsmål, derfor er vi nødt til at tage billeder ud af relationen Svarer på, og lade billeder selv lave en reference. Vi vælger at lade billeder referere direkte til enhederne tjekliste og spørgsmål. Nu kan vi tilføje adskillige billeder til et bestemt spørgsmål i en bestemt tjekliste uanset om der er svaret på spørgsmålene. relationen til hver enhed er en én til mange og fungerer ved at billeder har en fremmednøgle til hver.

Endelig lever vi også op til normalform 3.

Vi mener også at vi nu lever op til Boyce-code normalform om at hver entitet har attributter som identificerer de andre attributter inden for samme entitet.

**Oprettelse af Database**

På bilag [12] ses oprettelsen af databasen med de vigtigste egenskaber. Som planlagt i E/R diagrammet er alle tabeller som er understreget i E/R også oprettet med primærnøgler. Der er to relations tabeller, den ene har to fremmednøgler, mens den anden har tre fremmednøgler. Tabellen for billeder indeholder selv to fremmednøgler og behøver derved ikke en separat relations tabel.

### Operativsystemer (Simon)

**Android**

Android er et open-source operativsystem som baserer sig på Linux kernel som er open-source og ledig til alle. Android er udviklet til at virke på mindre elektroniske maskiner som tablets og mobiltelefoner. Android blev først sponsoreret af Google i år 2005, og i år 2008 var den første Android enhed udgivet.

Fordi at Android er open-source, kan alle der vil hente kilde-koden ned og ændre på den som de har lyst. Det betyder at forskellige producenters produkter ikke nødvendigvis har den samme version af android på deres produkter, selvom de er udkommet på tidspunkt.

Android bliver ofte udviklet ved brug af programmeringssprogene Java eller Kotlin.

Med en app lavet i Android kan man, få den lagt op på Google Play Store så alle kan hente appen ned på deres enhed.[[3]](#footnote-3)

Fra maj 2022 ligger Androids market share på 71,43%.[[4]](#footnote-4)

**iOS**

iOS bliver udviklet og udelukkende brugt af Apple i deres tablets og smartphones. iOS er ikke open-source, så du kan ikke lave dine egne ændringer for at tilpasse det til din app. Dette giver dem en vis kontrol over hvad for slags apps der kan laves.

Det er Apple der eksklusivt bestemmer hvilke apps der får lov til at komme på Apple Store.

Top programmeringssproget man bruger for at lave apps iOS bliver kaldt for Swift, men Java kan også bruges.

Fra maj 2022 ligger iOSs market share på 27,85%.[[5]](#footnote-5)

**Tråde (Threads)**

Vi har ikke brugt tråde i vores program, da vi ikke har set noget behov for at bruge det og har derfor heller ikke undersøgt nærmere hvordan de måske kunne implementeres i vores program.

# Programmering

### Email (Simon)

I SendEmail Klassen har vi alt der har med sendingen af emails at gøre.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse[[6]](#footnote-6)

På linje 16 declarere vi 3 EditText felter, som vi på linje 23-25 assigner til ID’et på de 3 EditText felter der bruges i activity-mail.xml filen, som er vores email activity.

På linje 27 initializer vi en Button, og forbinder den med knappen i activity-mail.xml filen.

På linje 28 står der hvad der skal ske, når man trykker på knappen der blev lavet på linje 27. Der har vi lavet en setOnClickListener, som vil udføre metoden kaldet sendMail() når knappen ’buttonSend’ bliver trykket på.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse[[7]](#footnote-7)

I sendMail() metoden på linje 37 initializer vi Stringen ’fileName’, som skal indeholde navnet på den .pdf fil vi skal tilføje til vores email. ’fileLocation’ variablet vil finde .pdf filen, ved hjælp af ’fileName’. På linje 39, skaber vi et Uri variabel ud fra ’fileLocation’ variablet, som vi kalder ’uriPath’. Uri bruges til at identificere hvilken resurse man bruger.

På linje 41 initializer vi et String variabel kaldet ’recipientList’, som får sin information fra ’mEditTextTo’ variablet.

På linje 42 skaber vi en datastruktur af typen array, og kalder det for ’recipients’. Dette recipients array vil tage Stringen ’recipientList’ og dele Stringen op hvor der findes et ’,’ (komma) tegn, hvilet den gør ved at tage brug af ’.split’. Ved at splitte Stringen ved ’,’ gør at brugeren kan indtaste mailen på flere modtagere end en.

På linjerne 43 og 44 initializer vi to Strings der henter deres information fra henholdsvis ’mEditTextSubject’ og ’mEditTextMessage’.

På linje 46 gør vi brug af Intent klassen, som gør det muligt at kommunikere med andre activities, og laver et objekt af den kaldet ’intent’. ’intent’ sætter vi til at være en ACTION\_SEND intent for at kunne sende emails.

Vi bruger nu vores ’intent’ objekt, og tilføjer mere til objektet ved brug af .putExtra. I .putExtra() fortæller vi programmet at det er en email vi har gang i og vi tilføjer vores String array af recipients. Det samme gør vi for vores String variabler ’subject’, ’message’ og vi tilføjer også en fil ved hjælp af EXTRA\_STREAM og vores ’uriPath’ vi lavede tidligere.

Til sidst sætter vi typen på vores intent og starter acitivity’en.

SendEmail klassen er ikke færdig, da den ikke virker perfekt endnu, og vi vil arbejde videre på den i de næste 2 uger op til eksamen.

### Design mønster (Kristian)

I klassen hvor vi forbinder til databasen bruger vi designmønsteret singleton for at skabe en stabil forbindelse som ikke er for resursekrævende. Designet består i at lave en forbindelse til databasen, og at sikre at der kun er denne ene forbindelse, uanset hvilken klasse som har brug for forbindelse til databasen.

For at sikre denne ene databaseforbindelse opretter vi klassen DbConnect med en constructor som er privat. En offentlig metode oprettes som først tjekker om konstruktøren er blevet kaldt. Er den ikke blevet kaldt, så er der ikke i forvejen oprettet forbindelse til databasen og DbConnect starter en forbindelse til databasen, men er konstruktøren i forvejen blevet kaldt så der er forbindelse i forvejen, benytter den denne. På den måde vil programmet kun have én forbindelse til databasen, det sikres at kommunikationen med databasen kommer i den rigtige rækkefølge, og at det er den rigtige forbindelse der

### Programmering af variabler (Kristian)

Her ses et problem, nemlig at der er overvældende mange variabler. Hver variabel indeholder et tal mellem -1 og 2 for ja nej irrelevant eller ikke udfyldt.

Et billede, der indeholder tekst

Automatisk genereret beskrivelseEt billede, der indeholder tekst, elektronik, skærmbillede, computer

Automatisk genereret beskrivelseJeg forsøger at løse problemet med et array, men for at holde styr på adskillige variabler med flere tal i hver, vil det blive et multidimensional array: int answer1[][]; i første [] skal henvise til nummeret på spørgsmålet, det anden [] skal henvise til svaret. Som det er nu, er det også nødvendigt at lave en metode for hver enkelt tjekboks hvilket vil være et uoverkommeligt stort arbejde. Jeg håber at et multidimentionalt array også vil løse den udfordring ved at gøre det muligt at lave en metode som bliver genbrugt.

Ved nærmere eftertanke vil det nok være bedre med et tredimensionelt array som tager hver gruppe, hvert spørgsmål og hvert svar. int answers[][][];



Det lykkedes at få 715linjers kode ned på 370linjer.

Metoden til at gemme valgene fra tekstboksene, er nu en metode som henter alle svar ved tryk på en gem knap. Men arket er større end skærmen og den kan ikke scrolle endnu. Derfor tilføjes scrolle funktionen i xml.

Scrolle funktionen kan indeholde et Linear layout som så indeholder de andre layouts. På den måde følger det træstrukturen i xml filen:

<ScrollView> -her startes scrollView

<LineaarLayout> -her starts linear layout

Her I linear layout kommer tekst og checkbokse, noget tekst ligger uden for scrolleviewet, det er for at den hele tiden skal være synlig mens man scroller på skærmen.

</LinearLayout> -linear layout sluttes først fordi den er child af scroll view

</ScrollView> -dernæst sluttes scroll view

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Et billede, der indeholder tekst

Automatisk genereret beskrivelseFor at gøre alle de mange spørgsmål nemmere at arbejde med vil jeg gerne have lagt dem i et todimensionalt array.

Lige nu ligger de i seperate strings. Først opretter jeg et todimensionelt arrayet list til tekst. Arraylist<Arraylist<String> questions = new Arraylist<arraylist<String>>(); fordelen ved array list er at man ikke behøver at definere størrelsen på forhånd.

Arraylisten genbruges til tekstfelterne fra xml og defineres som en liste med View i stedet for tekst. Formålet er senere at kunne autogenerere teksten til tekstfelterne i xml, samt at bruge arrayet til at fejlfinde på om checkboksene er udfyldt korrekt. Jeg vælger også at bruge arraylist fordi der er stor forskel på antallet af spørgsmål i hver gruppe af spørgsmål.

Et billede, der indeholder tekst, plade, skærmbillede

Automatisk genereret beskrivelseNår der gemmes, er hver enkelt information fra hver tjekboks skrevet ind manuelt, det tog tid og der er risiko for tastefejl. Med det nye todimensionelle array for Textview kan hele denne kode autogenereres. Det er cirka 90 linjer som ændres til to nestede loops. Der er dog det problem at arrayet answer fra verificationChart som den skal gemmes i er foruddefineret i størrelse og det vil helt sikker give problemer før eller side når loopene vil gennemløbe forskellige tomme elementer. Derfor må jeg ændre det til en todimensionel arrayList, det er egentligt et tredimensionelt array, men jeg synes det er på tide at forenkle lidt. Derfor vil spørgsmålene nu være fortløbende uanset hvilken gruppe i tjeklisten de tilhører.

Et billede, der indeholder tekst

Automatisk genereret beskrivelseNu er det tid til at forenkle koden som sætter teksten til xml tekst felterne. Lige nu er der 13 linjer, men der kommer flere. Det skal gerne kunne forenkles med en for løkke.

Et billede, der indeholder tekst

Automatisk genereret beskrivelseNu er koden nede på tre linjer. Desuden er den klar til at modtage flere linjer. Det vil den håndtere uden at skulle tilpasses, fordi den tilpasser loopet efter størrelsen på arraylisten.

Når checkfelterne skal gemmes, er det endnu en gang en fordel at få oprettet en liste. Man kan oprette an arrayList, men det vil være svært at holde overblikket når hvert spørgsmål har tre svarmuligheder. En simpel liste er mere overskuelig, fordi vi her har brug for en oversigt. Der er dog også en tredje mulighed og det er at lave en todimensionel arrayliste, så vil vi kunne søge på nummeret for spørgsmålet først og dernæst på svar valget. Det vil sikre at svarmulighederne ikke bliver blandet rundt på de forkerte spørgsmål.

Et billede, der indeholder tekst

Automatisk genereret beskrivelseHer bliver boolean listen answer fuldt med svarene fra xml som er gemt i listen check. Der bruges en while løkke til at sikre at der er indhold i listen check, inden der gemmes som boolean true eller false i answer listen fra klassen VerificationChart. Efter at jeg har kodet denne del af programmet, har vi lavet et E/R diagram og haft kundemøder. Og de overvejelser som der er gjort på E/R diagrammet samt kundesamtalerne betyder at koden ikke bare skal være mere enkel og autogenereret, men at den skal blive fuldt ud autogenereret, så antallet af spørgsmål kan variere efter behov.

### Database (Tim)

Grundet problemer med MySQL har jeg for denne opgave måtte hoppe tilbage til SQLite. Dette er gjort fordi jeg følte at jeg var under tidspres. Vi har ikke købt en online database da vi ikke pengene til dette. Men har lavet det så man kan ændre i en klasse for at få det til at virke.

Vi har lavet en Klasse som hedder DatabaseHelper som vil indeholde alt der omhandler databasen.

Når Appen Registrer en ny bruger for første gang kontrollere den om der er en database først. Derefter laver den en database to mobilen som hedder users.db

Når man har udfyld alle de registret felter bliver og har trykket på Registrer køre den addUser metoden. Først åbner man for databasen i en skrivebar tilstand som er skrevet med den anden linje på billedet under. Så fortæller man at man vil indskrive de følgende info ind i databasen: username, e-mail og password. Og så returnere man en boolean der fortæller om det er lykkes eller ej.Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Der bliver også kontrolleret i databasen om brugeren findes før den førnævnte metode. Dette bliver gjort i metoden checkUser. Først åbner man for databasen i en skrivebar tilstand som er skrevet med den anden linje på billedet under. Så måler den det indtastet username imod dem der er i databasen og sender true eller false tilbage afhængigt af svaret.Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Under login vil der blive kontrolleret i databasen om username og password findes der inde samt om de er i overens med databasen

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

For videreudvikling af appen ville der blive lavet eller købt en online database som opgaverne og brugerne ville blive gemt på.

Der vil også blive skiftet SQLite ud med en anden SQL som virker bedre med online databaser.

### Login (Tim)

Jeg har grundet en masse fejl måtte lave Login om en del antal gange og fandt til sidst en måde som virket.

Jeg har dog lavet den fejl at jeg har lavet det hele i activity’s og det bliver ændret inden den endelige udgivelse.

Både Login og Registration har hver deres egen Klasse men de har også en til fælles som er DatabaseHelper

MainActivity er min login Klasse.   
Den har de muligheder at den kan enten logge dig ind hvis du allerede har en konto og har udfyldt felterne, eller den kan sende dig over til en Registrations klasse som kan lave en konto.

Dette sker ved at aktiverer diverse Buttons.

På linje 60 har jeg lavet en setOnClickListener, som vil udføre metoden onClick der vil skifte side fra Login til registration.

Et billede, der indeholder tekst

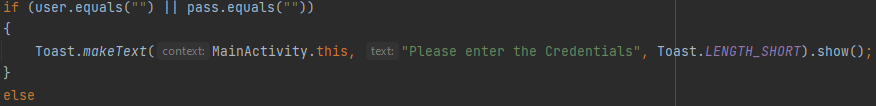
Automatisk genereret beskrivelse

Dette viser eksemplet på skiftet over til Registrations klassen.

I login er der kun brugt 2 strings til at kunne lave login.  
String user og password.

Jeg kunne godt tænke mig for fremtiden af tilføje mailen til login.

Jeg har nu lavet en masse IF statesments som også har indført nogle Toasts til hvad mangler der mangler for at kunne logge ind.

Dette eksempelt viser at hvis der mangler at blive udfyldt et af felterne vil den lave en Toast og ikke rykke videre til resten af appen. 

Derefter kontrollere den databasen om de indtastet informationer passer med hvad der står i databasen.  
Afhængigt af om det passer eller ikke passer returnere den en boolean værdi som sender dig videre ind i appen eller laver en Toast om at de indtastet informationer ikke passer med hvad der er i databasen.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Fremtidige planer for appen er at lave 2 slags kontoer for appen. En app for kontoret som kan lave opgaver og sende dem til sine ansatte, og en medarbejder konto som kan have forskellige muligheder givet fra kontoret.

Der kan muligvis blive indføre ordrestyrring i programmet hvis der er ønsker om det, og en god PC-version.

Ændre at jeg har lavet dem på ren Activity’s til fragments. Så kan man gemme en god bundlinje til at rykke rundt i appen med. Det vil gøre appen mere brugervenlig samt visuel mere tiltagende.

Der skal også læres mere om hvordan man kryptere informationer så de private informationer som kunden indtaster, ikke bliver taget så nemt. Ser helst gerne at det bliver så stærkt beskyttet at det ikke er til hacke for en uden meget erfaring.

### Registration (Tim)

Registration bliver suppleret meget af DatabaseHelper klassen.

Et billede, der indeholder tekst

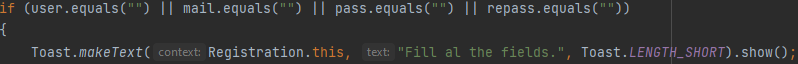
Automatisk genereret beskrivelseI Registration er der brug for lidt mere information så der er lavet 4 Strings.  
String User, e-mail, password og retypepassword.

Et billede, der indeholder tekst

Automatisk genereret beskrivelsePå linje 60 har jeg lavet en setOnClickListener, som vil udføre metoden onClick der vil skifte side fra Login til registration.

For at knapper på skærmen skal virke skal man huske at aktiverer dem i koden

Registrationen er bygget med IF statements og Toasts. Hvis en af statements fejler sender den en Toast og du kommer ikke videre.

Den første er en kontrol om alle felterne er udfyldt.

Derefter en kontrol om kodeordet passer med gentastningen af kodeordet, hvis kodeordene passer kommer du videre. Hvis de ikke passer for du en Toast for siger at dine kodeord ikke passer.

Så kommer kontrollen af brugerens allerede er i databasen. Afhængigt af om der er en bruger med det samme navn eller ikke er. Returnere den en boolean værdi som går videre til at laver din konto eller laver en Toast om at de indtastet informationer ikke passer med hvad der er i databasen.

Så bliver kontoen lavet og tilføjet til databasen via DatabaseHelper Klassens metode addUser().  
Hvis det skulle fejle af en ukendt grundt vil den Lave en Toast der fortæller at du fejlede.Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Registration af brugeren vil blive udvidet en del da der skal underskrift på hvem der har lavet kontrollen.

### Generering af pdf-fil (Kristian)

For at danne tjekskemaet som pdf benytter vi iText. iText biblioteket importeres i programmet så at vi med enkel kode kan kalde iText filerne og danne pdf’erne.

I AndroidManifest.xml gives der lov til at programmet må skrive til anden lagerplads (External storage). Derefter tilføjes afhængighed (dependencies) i filen build.gradle. blandt andet tilføjes web-adressen så android studio nu kan synkronisere iText biblioteket med versionen itext7-core:7.2.2.

For at gøre det muligt for programmet at gemme en pdf på den enhed den kører på skal den have en tilladelse. Det gøres ved i filen AndroidManifest at tilføje linjen: android:allowBackup="true" , hvorpå den ved første opstart af programmet vil bede om tilladelsen. Når tilladelsen er givet, vil der kunne gemmes pdf’er på enheden.

# Systemudvikling

## Kravindsamling og -specifikation

### Use cases (Simon)

* **Casual use case**

Vi har lavet og benyttet en casual use case i starten af vores projekt, til at få et hurtigt og nemt overblik over hvordan en bruger ville interagere med vores app.

* **Formelle use case**

En formel use case er meget mere detaljeret, og gør det muligt at få afklaret flere ting. Skabelonen vi har brugt til at lave vores formelle use case ser sådanne ud:

* **Use case name**
* **Scope:** Appen eller systemet der er under udvikling.
* **Primary actor:** Hvem kommer til at bruge systemet.
* **Level:** Kan enten for subfunction eller user goal.
* **Description**
* **Stakeholders and interests:** Kan være firmaet der betaler for udviklingen af appen
* **Preconditions:** Her beskrives hvilke forudsætninger der skal være for at systemet kan bruges.
* **Succes guarantee:** Hvad skal der ske for at use casen kan siges at have været en succes
* **Main succes scenario:** Det typiske scenarie der forekommer for at få et succes guarantee.
* **Extensions:** En beskrivelse af andre mulige udfald af use casen.
* **Special requirements:** Specielle krav for at use casen kan forløbe.
* **Frequency of Occurrence:** Beskriver hvor tit denne use case forekommer.
* **Miscellaneous:** Beskriver ting her der ikke hører til nogen af de andre felter.

## Object-Oriented Analysis (OOA)

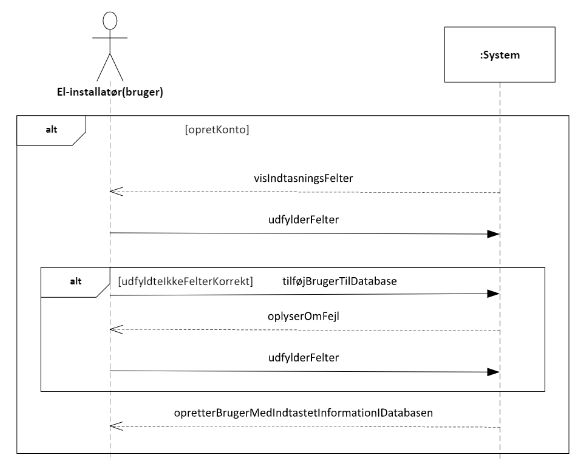
### System sekvens diagrammer (SSD) (Simon):

Et system sekvens diagram er et artefakt, der giver et overskueligt kig ind i hvordan programmets system reagerer (output) baseret på hvad brugeren af programmet giver af input.

I en SSD starter du ud med at putte en ’bruger’ ind. Ud fra denne bruger skal du lave en lifeline, som du hænger på brugeren, lifelinen er en linje som altid vil gå ned af. Denne lifeline vil illustrere tiden i dit program. Eventsne i en SSD skal skrives ud fra en use-case man tidligere har lavet, og følge dens opstilling.

I vores gruppes SSD (udsnit ses nedenunder), vil brugeren skulle begynde ud med at oprette en konto. Brugeren starter med at trykke på ’opret konto’ knappen og systemet viser siden hvor du vil lave din konto. Systemet viser de forskellige indtastnings felter og brugeren vil derefter udfylde felterne og trykke på ’opret konto’ knappen.

Svar eller beskeder fra systemet i en SSD illustreres som en stiplet pil, der peger mod brugerens lifeline. En besked eller request fra brugeren til systemet, illustreres med en pil der peger mod systemets lifeline. Et eksempel på dette kan ses i udsnittet. Systemet viser de forskellige indtastnings felter og der peges en stiplet pil mod brugerens lifeline. Brugeren udfylder felterne og trykker på ’opret konto’ knappen, og der peges en pil mod systemets lifeline.

[[8]](#footnote-8)

I udsnittet er det hele indkredset i en boks, hvor der i hjørnet står ’alt’. ’Alt’ står for alternativ, og det betyder at alt inde i boksen er et alternativ, hvilket betyder at for at bruge appen, behøver brugeren ikke nødvendigvis at oprette en konto.

Inde i den første ’alt’ boks, findes der endnu en ’alt’ boks. Den beskriver hvad der sker hvis brugeren indtaster noget forkert i forsøget på at oprette en konto. Hvis brugeren indtaster noget forkert og trykker på ’opret konto’ knappen, vil systemet oplyse at der er sket en indtastnings fejl og at brugeren skal forsøge at indtaste sine kontooplysninger igen. Efter at brugeren har indtastet sine kontooplysninger på ny, trykker brugeren igen på ’opret konto’ knappen og denne gang er alt i orden så systemet opretter brugeren i databasen.

Et billede, der indeholder bord

Automatisk genereret beskrivelse[[9]](#footnote-9)

I udsnittet ovenover fortsætter er vi nået til punktet i vores program hvor brugeren vil logge ind. Brugeren er på forsiden og trykker på ’log ind’ knappen. Systemet svarer ved at vise loginskærmen. Brugeren udfylder loginfelterne og trykker på ’log ind’ knappen. Der kommer nu en ’alt’ boks hvor der vises et alternativ til et succesfuldt login. Brugeren har indtastet sine loginoplysninger forkert og systemet svarer med en fejlmeddelelse, og beder brugeren forsøge at indtaste sine loginoplysninger igen. Brugeren forsøger at indtaste de korrekte loginoplysninger og denne gang er brugeren succesfuld, og systemet svarer ved at logge brugeren ind, og viser skema siden hvor brugerens igangværende skemaer er, og hvor brugeren kan skabe et nyt skema.

Et billede, der indeholder bord

Automatisk genereret beskrivelse[[10]](#footnote-10)

I dette udsnit viser SSD’en nogen alternative bokse.

Den første ’alt’ boks beskriver hvad der sker hvis brugeren vælger et eksisterende skema. Brugeren vælger at udfylde et eksisterende skema. Systemet svarer med at vise det valgte skema. Brugeren udfylder skemaet.

Den anden ’alt’ boks beskriver hvad der sker hvis brugeren vælger at oprette et nyt skema. Brugeren trykker på ’opret nyt skema’ knappen. Systemet svarer ved at vise et blankt skema. Brugeren udfylder skemaet.

Et billede, der indeholder bord

Automatisk genereret beskrivelse[[11]](#footnote-11)

Dette udsnit af SSD’en beskriver tilføjningen af billeder og færdiggøringen af skemaet.

I dette udsnit findes der en ’opt’ boks. ’Opt’ står for ’optional’, hvilket vil sige det er en valgfri ting at tilføje billeder til ens skema, og det er ikke nødvendigt for at færdiggøre sit skema.

Brugeren trykker på ’tilføj billede’ knappen og vælger de billeder der skal tilføjes. Systemet svarer ved at tilføje billederne til skemaet. Brugeren trykker på færdig knappen og systemet svarer ved at skabe en .pdf-fil af skemaet.

Et billede, der indeholder bord

Automatisk genereret beskrivelse[[12]](#footnote-12)

Det sidste udsnit af SSD’en beskriver oprettelsen af en mail og sending af mail.

Brugeren trykker på ’opret mail’ knappen. Systemet svarer med at vise email activity siden, og en liste over potentielle modtagere. Der er nu en ’opt’ boks, i denne boks vælger brugeren at vælge en modtager fra listen over modtagere.

I den anden ’opt’ boks indtaster brugeren selv emailen på modtageren. Brugeren fortsætter ved at indtaste titlen på sin mail, indtaste en besked og trykker på ’send mail’ knappen. Systemet svarer ved at tilføje skemaet nu i en .pdf fil, og videre sender brugeren til den valgt email app, som i dette scenarie er Gmail, hvor brugeren vil færdiggøre sin email og sende.

### Domænemodel (Simon)

Vi har lavet en domænemodel der illustrerer de konceptuelle klasser i vores program.

Domænemodellen kan ses i bilag 1. I Registration klassen vil registreringen af en bruger tage sted. Den er forbundet til Logind klassen hvor at brugeren skal indtaste de loginoplysninger brugeren skabte ved registreringen. Med en multiplicitet på 0..1 ved Registration og 1 ved Logind fordi en bruger kun kan lave 1 konto, men behøver ikke at lave en konto hver gang brugeren bruger appen, og 1 ved Logind klassen fordi at brugeren altid skal bruge information fra Registration klassen for at kunne login.

Logind klassen er også forbundet med Hjemmeskærm klassen, der vises efter man er logget ind. Multipliciteten ligger på 1 begge veje her, da man kun kan logge ind 1 gang og du kun vil kunne gå fra loginskærmen til hjemmeskærmen 1 gang pr login.

Hjemmeskærmen er forbundet til flere klasser, en af dem værende Konto klassen. I Konto klassen vil du kunne ændre dine kontooplysninger. Fra hjemmeskærmen er der 1 instans, og 0-1 mod Konto klassen, da det ikke er nødvendigt at ændre sine kontooplysninger. Konto klassen er også forbundet med Registration klassen, fra Konto klassen med 1 instans, og en multiplicitet på 0-1 hen mod Registration klassen da brugeren ikke behøver at ændre sine kontooplysninger fordi brugeren gik ind på siden hvor man ændrer sine oplysninger.

Hjemmeskærmen er forbundet til LabnyOrdre Klassen, med en instans på 1 ved Hjemmeskærmen, og en multiplicitet på 1..\* (1-mange) mod LavNyOrdre, da en bruger kan finde på at lave 1 eller flere ordrer.

Hjemmeskærmen er også forbundet med Ordre klassen, med 1-1 multiplicitet da brugeren kun kan trykke på én i gang værende ordre afgangen.

LavNyOrdre klassen er også forbundet med Ordre klassen med 1-1 multiplicitet, da man kun kan genere 1 ordre ad gangen.

Til sidst er Ordre klassen forbundet med AfslutOrdre klassen som sørger for at sende mailen afsted med .pdf filen. De to klasser er forbundet med 1-1 multiplicitet da man kun kan afslutte 1 ordre adgangen.

Vores domænemodel var noget af det første vi lavede, og der har været ændringer i vores program siden vi lavede den, så derfor vil domænemodellen ikke være helt opdateret.

## Object-Oriented Design (OOD)

### Klassediagram (Tim)

Klassediagrammet har fire vigtige klasser som er MainActivity VerificationChart og User og DbConnect.

MainActivity fungerer som en controller for userinterfacet. Ved login aktivere MainActivity User klassen til at hente data fra Databasen om brugeren der fortæller om adgang til programmet og anden data fra databasen.

Tjeklisterne håndteres ved hjælp af klassen VerificationChart. Ved oprettelse af et objekt ud fra VerificationChart sikres en god håndtering af data til et bestemt tjekskema. Dataen hentes og gemmes i databasen ved hjælp af forbindelsen via DbConnect. DbConnect er en singelton som kun giver én forbindelse til Databasen.

PdfGenerator aktiveres fra MainActivity, men henter data fra VerificationChar til at danne pdf’er. SendEmail klassen gennrerer en standard e-mail som kan sendes til en bruger som allerede er oprettet

### Sekvensdiagram (Kristian)

Et billede, der indeholder bord

Automatisk genereret beskrivelseUdarbejdelsen af sekvensdiagrammet har til formål at give en oversigt over en proces i programmet. Sekvensdiagrammet inkluderer flere objekter, og de viser også hvor vi er i sekvensen tidsmæssigt. Metoderne i tre klasser i programmet er begyndt at blive uoverskuelige, og derfor laver jeg nu et sekvensdiagram som jeg vil kunne arbejde ud fra. Jeg vil gerne have at diagrammet indeholder sekvensen for at der hentes tekst, at svar gemmes og at der genereres en pdf.

Klassen VerificationChart holder styr på data som har at gøre med tjeklisten. Som start i vores proces gemmes der i klassen VerificationChart, men senere vil VerificationChart få sine data fra en database. Databasen lukkes gentagne gange for at sikre at der ikke opstår konflikter når andre klasser laver forbindelse til samme database. Er man sikker på at der ikke vil opstå konflikt, vil databaseforbindelsen kunne skabes én gang og genbruges, her er den sikre løsning valgt.

Ved opstart af en tjekliste bliver der genereret et nyt objekt vc af VerificationChart. Fra vc hentes oplysninger i form af tekst som bruges til at generere tekster i det grafiske userinterface, der hentes også svar samt info om billeder hvis tjeklisten allerede eksisterer i databasen. Vc tager sig af at tjekke om tjeklisten allerede eksisterer i databasen, og er på diagrammet indrammet med en boks markeret med ALT for alternativ, hvis tjeklisten ikke eksisterer i databasen, vælges alternativet som er at generere en standart formular til indtastning. Standart formularen er endnu ikke gemt i databasen.

Når metoden save kaldes gemmes der information i det todimensionelle array check som allerede er defineret i klassen MainActivity. Informationen hentes fra det grafiske user interface, og MainActivity fungere således som en controller for user interfacet. Efter at informationen er gemt i check kopieres den over i vc som allerede er oprettet og gemmes her i det todimensionelle array answer. Man kunne have ladet vc selv gemme den information direkte, men for at give rum for fremtidige ændringer holdes kommunikationen med det grafiske userinterface og MainActivity adskilt fra vc. Skifter man til at andet grafisk userinterface som kræver oversættelse eller tilretning af informationen, vil det være naturligt at ændre det her i MainActivity inden det gemmes i vc.

Et billede, der indeholder bord

Automatisk genereret beskrivelseNår funktionen send kaldes, oprettes et objekt af klassen Pdf og navngives pdf. i pdf kaldes metoden sendPdf. I første omgang var sekvensen planlagt til at hente information i databasen, som vist på det lille billede. Med det er der ikke nogen grund til når vc allerede er opdateret først i sekvensen og igen med save funktionen. Nu kan vc informationen kan hentes direkte. Var pdf ikke aktiveret i denne sekvens kunne det være nødvendigt at hente informationen fra databasen. genneratePdf går ikke til mainActivity fordi pdf gemmes direkte på enheden. Senere tilføjes der måske en besked til MainActivity om at pdf’en er genereret, men for nu undlades den besked.

## Kvalitetssikring

### Brugertest med projektstiller (Kristian)

Ved første kundemøde er vi blevet informeret om kundens problemstilling og ønsker.

Vi har fået at vide at elinstallatøren er ansvarlig for at dokumentere hvad der er lavet og udført i forhold til gældende lovgivning. En hjælp til det er sikkerhedsstyrelsens skema som er lagt ud som en pdf-fil. Skemaet er kun vejledende. Dokumenteringen skal opbevares i op til 5 år. Dokumentet bruges til kontrol i små kontorer og boliger. En bolig kan have flere lokaler. Kontrollen kan vare i et par dage, og der kan være brug for rumopdeling. Det er ikke sikkert at skemaet bliver udfyldt på en gang, men man håber på at medarbejderen kommer igen dagen efter og fortsætter hvor han slap. Vi skal også regne med at en medarbejder måske udfylder skemaet nogle dage efter. Desuden kan der være brug for løbende kontrol i løbet af projektet. Det er en fordel at kunne tilføje billeder. Nogle medarbejdere har tablets, men det kan der ikke garanteres for, der kan garanteres for at alle har mobiltelefoner.

De efterfølgende møder er aftalt:

Tirsdag 17/5 12:30

Onsdag 25/5 12:30

Fredag 3/6 12:30

Ved andet kundemøde blev udfyldnings formularen er vist til kunden. Han bemærkede at radioknapperne på forhånd er valgt, hvilket ikke er hensigtsmæssigt. Der skal være mulighed for at spørgsmålet helt kan undlades. Vi har også fået forklaret at mens at installatøren har ansvaret kan hvem som helst han ønsker udfylde skemaet. Vi fik også præciseret at billederne skal forbindes til hver spørgsmål som de hører til.

Ved tredje kundemøde kunne vi vise hvordan tjekboksene nu ikke er forud valgt, hvilket var tilfredsstillende. På grund af tidshorisonten var vi nødt til at spørge ind til hvilken del af programmet som er vigtigst at få lavet færdig. Og vi aftalte at prioritere side to i tjekskemaet som indeholder målinger frem for at tilføje muligheden med at tilføje billeder.

På fjerde kundemøde måtte vi fortælle kunden at vi stadig arbejder på programmet, men at der grafisk ikke er nogle ændringer siden sidst. Vi glæder os til at have et færdigt produkt.

### Brugergrænseflade test (Kristian)

For at udføre test af brugergrænsefladen startes applikationen.

Visuelt testes der at teksten fra klassen VerificationChart bliver vist.

Først ud kommenteres metodekaldet setText i metoden onCreate i klassen MainActivity. Programmet køres og der vises teksten ”Afventer”. Dernæst fjernes ud kommenteringen så at metodekaldet til setText vil blive aktiveret ved opstart af siden. Programmet genstartes og teksten er nu udskiftet til teksterne fra VerificationChart. Hermed kan jeg konkludere at arraylisten questions i VerificationChart er i funktion, og at construkteren i VerificationChart får fyldt teksterne i questions arraylisten. Metoden setText i klassen MainActivity fungerer også og er blevet kaldt ved opstart.

Det testes også at besvarelser fra brugeren bliver opsamlet og gemt.

Jeg vil sikre at dataene bliver gemt i det todimensionelle array answer i VerificcationChart. (i øjeblikket er det et todimensionelt array, men på grund af kundesamtalerne og E/R diagrammet skal den snart laves om til en autogenereret liste som for eksempel en Arraylist.) For at teste, koder jeg et loop sidst i metoden save i MainActivity som udskriver alle data fra answer til terminalen. Koden får lov at blive stående men ud kommenteret så det er nemt at teste igen på et senere tidspunkt. Inden udgivelse af programmet skal alle tjekbokse testes, det kan eventuelt være at vi senere har fået autogenereret spørgsmålene og tjekboksene, og i så fald vil vi til den tid kunne nøjes med at teste et spørgsmål og dets fire svar muligheder, nemlig: ja, nej, ikke relevant og ikke udfyldt.

Et billede, der indeholder tekst

Automatisk genereret beskrivelseNu startes programmet og jeg vælger at tjekke af i ja på spørgsmål et, og nej i spørgsmål to og ikke relevant i spørgsmål tre. Jeg trykker gem og værdierne udskrives til programmet.

På billedet fra terminalen ses svarene på første spørgsmål.

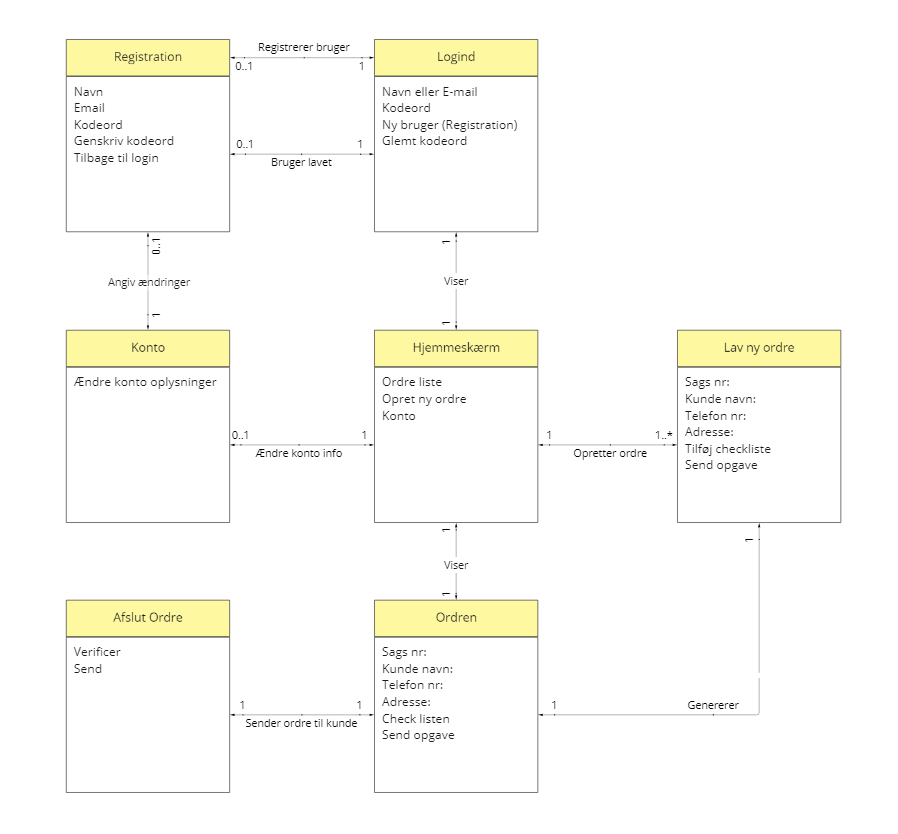
Ja er true, nej er false, irellevant er false. Der er altså markeret ja i første spørgsmål, og det er sandt. Testen viser at dataene bliver gemt rigtigt, og at de bliver modtaget fra brugerfladen fra xml filen håndteret i controler klassen som her er kaldt MainActivity og gemt i klassen VerificationChart som er oprette med det formål at håndtere data fra tjekskemaet.

# Konklusion (Simon)

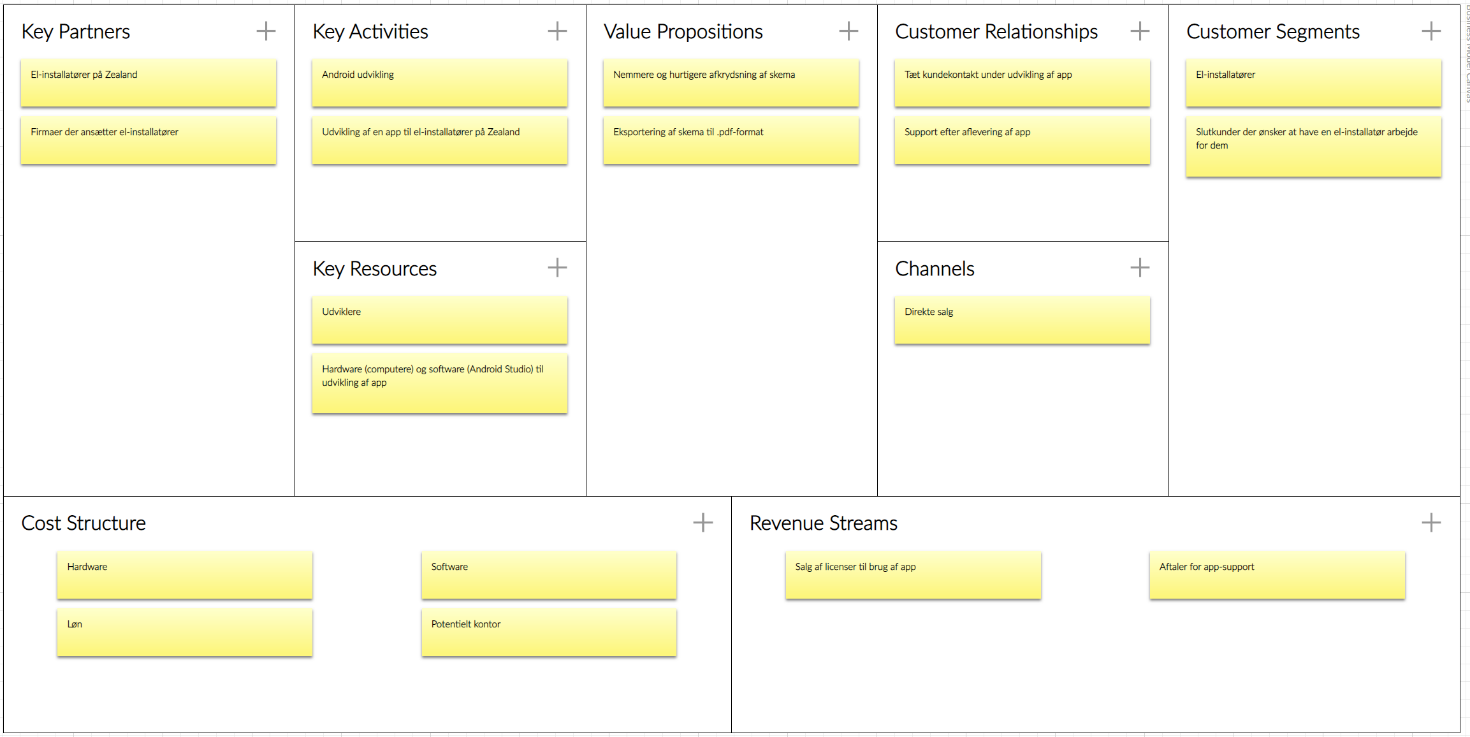
Vi startede ud med kundemødet og fik styr på de krav kunden havde til vores app. Derefter lavede vi en use case og domæne model, og ud fra det begyndte vi at arbejde på vores første kode. Vi kiggede på forskellige virksomheder, og fandt en virksomhed der havde et produkt der mindede om vores eget, ud fra det kunne vi konkludere at der var en efterspørgsmål på vores produkt udover Zealand. Efter vi fik lavet vores diagrammer færdige, gik vi i gang med kun at kode, og vi mener vi er kommet okay til måls, men der er stadig ting der skal laves i tiden op til eksamensdatoen. Det har været en stor udfordring, og vi har ikke kunne nå at lave alt vi ønskede, men projektet har tvunget os til at bruge alt vi har lært og vi har lært meget om hvordan man arbejder sammen i et projekt.

# Bilag

Bilag 1: Domænemodel



Bilag 2: BMC



Bilag 3: Investeringsbudget

Et billede, der indeholder bord

Automatisk genereret beskrivelse

Bilag 4: Risiko analyse



Bilag 5: System Sekvens DiagramEt billede, der indeholder bord

Automatisk genereret beskrivelse Et billede, der indeholder bord

Automatisk genereret beskrivelse Et billede, der indeholder bord

Automatisk genereret beskrivelse

Bilag 6: Casual usecase

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Bilag 7: Formelle usecase

Et billede, der indeholder tekst, skærmbillede, monitor, sort

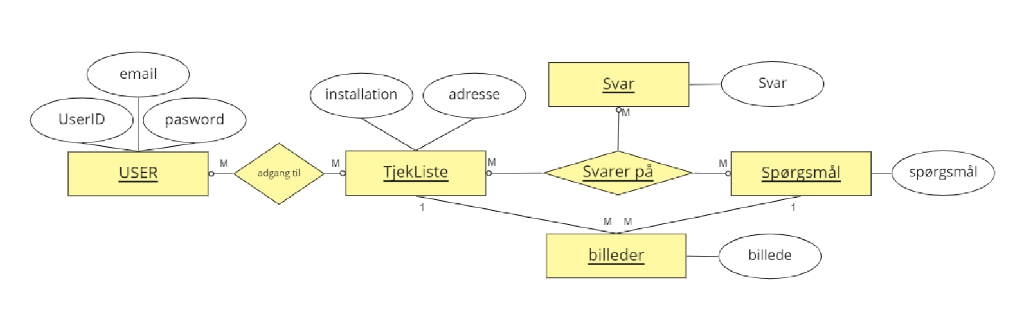
Automatisk genereret beskrivelse

Bilag 7: Sekvensdiagram

Et billede, der indeholder bord

Automatisk genereret beskrivelse

Bilag 8: E/R Diagram



Bilag 9: E/R Diagram 1. iteration

Et billede, der indeholder tekst, whiteboard

Automatisk genereret beskrivelse

Bilag 10: Klasse diagram

|  |  |
| --- | --- |
| Klassenavn | User |
| Documentation: | |
| Attributes: +onCreate() | |
| MainActivity | Efter log ind |
| DbConnection | Registrer en ny bruger |
| String | Brugernavn |
| String | E-mail |
| String | Kodeord |

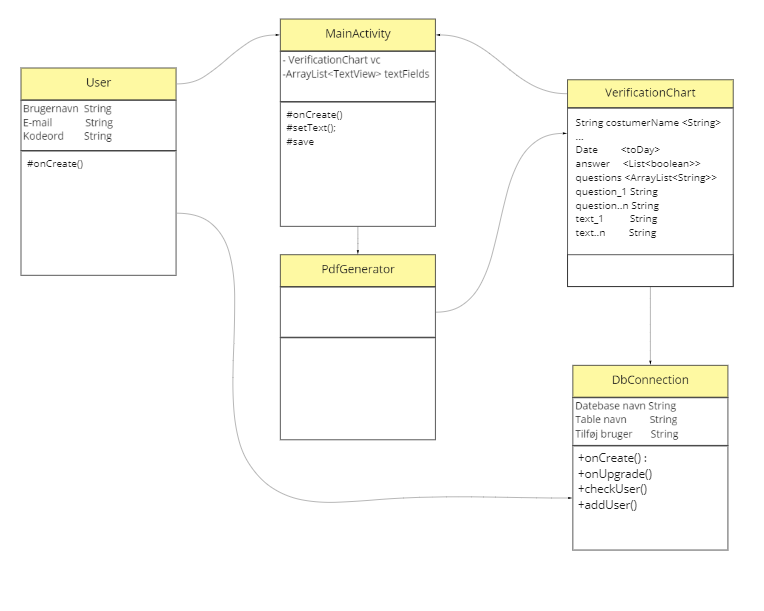
|  |  |
| --- | --- |
| Klassenavn | DbConnection |
| Documentation: | |
| Attributes: +onCreate() +onUpgrade() +checkUser() +addUser() | |
| User | Registration af ny bruger |
| VerificationChart | Færdig gjort checkliste gemt og sendt |
| String | Database navn |
| String | Table navn |
| String | Tilføj bruger |

|  |  |
| --- | --- |
| Klassenavn | MainActivity |
| Documentation: | |
| Attributes: #onCreate() #setText() #save() | |
| User | Modtaget bruger info fra |
| VerificationChart | Lavet en ny eller valgt en opgave |
| PdfGenerator | Færdiggjort opgaven og gemt |
|  | VerificationChart vc |
|  | ArrayList<TextView>textFields |

|  |  |
| --- | --- |
| Klassenavn | PdfGenerator |
| Documentation: | |
| Attributes: | |
| VerificationChart | Konstruktionen af en opgave. |
| MainActivity | For spøgelse af at lave en ny opgave |

|  |  |
| --- | --- |
| Klassenavn | VerificationChart |
| Documentation: | |
| Attributes: | |
| MainActivity | Efter at havde valgt en opgave |
| DbConnection | Når opgaven er færdig og gemt inden den bliver sendt. |
| PdfGenerator | Konstruktionen af en opgave. |
| String | CostumerName |
| String | Question 1 |
| String | Question…n |
| String | Text 1 |
| String | Text…n |
| String ArrayList | Questions |
| Boolean | Answer |
| String | Date |

Bilag 11: Klasse diagram (Tim, Simon, Kristian)



Bilag 12: Oprettelse af tabeller i database ved hjælp af SQLite

Et billede, der indeholder tekst

Automatisk genereret beskrivelseEt billede, der indeholder tekst

Automatisk genereret beskrivelseEt billede, der indeholder tekst

Automatisk genereret beskrivelse

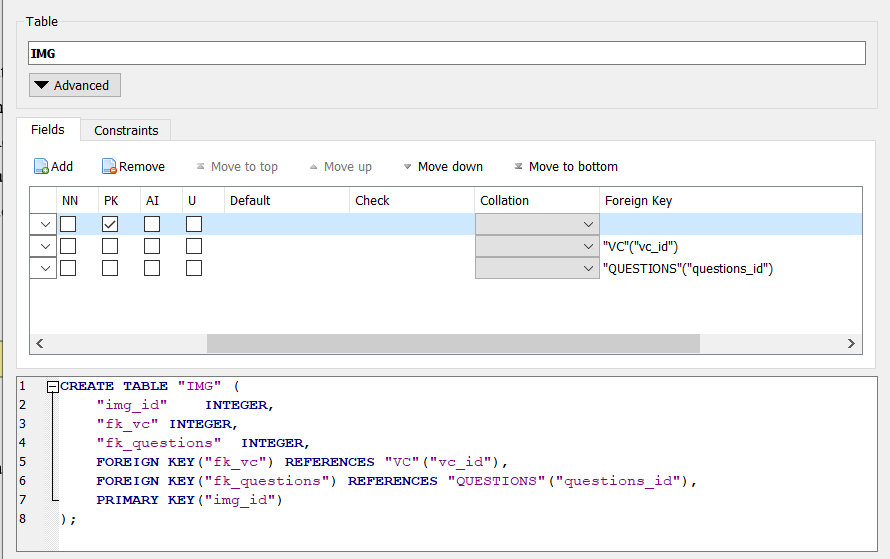
Et billede, der indeholder tekst

Automatisk genereret beskrivelseEt billede, der indeholder tekst

Automatisk genereret beskrivelse

Et billede, der indeholder tekst

Automatisk genereret beskrivelse



Udprint ved hjælp af SQLite, af oversigt over database

Et billede, der indeholder bord

Automatisk genereret beskrivelse

1. Github projekt, <https://github.com/Kristian-j-k/Elinstallator_hold4/projects/1> [↑](#footnote-ref-1)
2. Udsnit af risikoanalysen, viser analysen af sygdom blandt udviklere. [↑](#footnote-ref-2)
3. Webside: <https://www.elprocus.com/what-is-android-introduction-features-applications/> [↑](#footnote-ref-3)
4. Webside: <https://gs.statcounter.com/os-market-share/mobile/worldwide> [↑](#footnote-ref-4)
5. Webside: <https://gs.statcounter.com/os-market-share/mobile/worldwide> [↑](#footnote-ref-5)
6. Screenshot: af øverste del af SendEmail klassen, viser variabler. [↑](#footnote-ref-6)
7. Screenshot: af nederste del af SendEmail klassen, viser sendMail() metoden. [↑](#footnote-ref-7)
8. Screenshot: Udsnit af SSD OpretNytSkema, illustrerer hvordan systemet opretter en bruger i databasen. [↑](#footnote-ref-8)
9. Screenshot: Udsnit af SSD OpretNytSkema, illustrerer hvordan systemet opfører sig når brugeren forsøger at logge ind. [↑](#footnote-ref-9)
10. Screenshot: Udsnit af SSD OpretNytSkema, illustrerer alternative scenarier ved oprettelsen af et skema. [↑](#footnote-ref-10)
11. Screenshot: Udsnit af SSD OpretNytSkema, illustrerer tilføjning af billeder og skabningen af en .pdf fil. [↑](#footnote-ref-11)
12. Screenshot: Udsnit af SSD OpretNytSkema, illustrerer hvordan der oprettes en mail, vælges mail modtager og videre sending til Gmail app. [↑](#footnote-ref-12)