OBLIG 2

ROBERT LARSEN

APP UTVIKLING

S349967

20.11.2022

Innhold

Innledning

Spillet følger en følgende kravspesifikasjon gitt i oblig2, apputvikling, vår 2022:

- Legge til nye kontakte
- Sette kontakter
- Endre kontakt informasjon
- Legge til nye avtaler
- Slette avtaler
- Endre avtaler
- Notification om avtaler til bruker, som ikke kan slås av.
- En kan ta av og på servicen som gjør at en får sendt sms og notifikasjoner
- Tidspunktet når notifikasjoner og sms blir sendt, er satt i sharedprefrence.
- Hvis ingen beskjed er gitt i en avtale, brukes default beskjed gitt i shared prefrence.
- Content provider

Oppgaven går ut på å lage en applikasjon som gir brukere mulighet til å organisere avtaler med kontakter. Det vil si at en kan opprette, slette og oppdatere avtaler og kontakter, samt at denne informasjonen deles videre til kontaktene. Måten dette skulle gjøres på, var å gi alle kontakter på den dagen en har avtale, en sms. Videre skulle det gis notifikasjon til brukeren, om at det er avtaler i dag. Utover dette skulle det lages contentprovider, slik at tredjeparts programvare får tilgang til disse dataene – som eventuelt kan dele informasjonen på andre måter.

Innledningsvis om design valg

Både Nielsen og Tognazzini var opptatt av at det en gjør på datamaskinen, skal likne mest mulig på tilsvarende aktiviteter i virkeligheten. [6]

Ved å ta utgangspunkt i hvordan ting kan gjøres mest effektivt med vanlig papir, så vil en enklere kunne se at det er lurt å bare utsette personen for informasjon som er relevant for søket – når personen faktisk søker etter riktig informasjon.

Både Tognazzini og Nilsen var også opptatt av at noe skulle være effektivt og enkelt, dvs irrelevant informasjon oppleves som støy. [6]Kun informasjon som er relevant for de arbeidsoppgaver en nå utfører, skal være tilstede. Også dette vil en enklere kunne se, hvis en kun jobber med papir -dvs å gjøre noe i den virkelige verdenen. Årsaken til at det er lettere å se, er at kostnadene ved å gjøre feil er mye større – en er da mer tilbøyelig med å unngå å gjøre dårlige design valg.

For både kontakter og avtaler får en liste av avtaler/kontakter først, slik at en fortere finner frem til riktig avtale/kontakt. Deretter får en mulighet til å gå til selve kontakten/avtalen, hvor en kan endre og slette. For avtale vil en i tillegg kunne legge til kontakter, som gjøres gjennom en seperat aktivitet.

Videre er det laget tilbakemeldinger til brukeren på at noe er lagret, endret, slettet og at brukeren har skrevet feil inn i et input felt. Dette er med på å støtte opp om Nilesne prinsipp om å gjenkjenne,

diagnotisere og å komme seg tilbake fra errors[6]. Videre er det med på å støtte opp under prinsippet om synlighet av systemets tilstand [6]

At lagre, endre og mest mulig er laget i egne aktiviteter tar altså utgangspunkt i Nielsens prinsipper om minimalistisk design og Nilesens prinsipp om at det skal være samsvar mellom virkeligheten og det som skjer på datamaskinen.

Generelt om lister

Å holde rede på informasjon er alltid lurt, spesielt med tanke på informasjon som er vanskelig å huske.

Når en har mye informasjon, er et av målene at en skal kunne finne igjen til informasjonen raskt. Da vil en liste over det en leter etter, være en måte å legge til rette for et slikt søk. Hva personen søker etter kan ha betydning for hvilke informasjon som er relevant å vise i listen. Kanskje personen trenger å vite egenskaper og attributter for hver ting i listen, for å gjøre en avgjørelse om at en har funnet fram til riktig ting.

For å effektivisere søk kan en sortere med hensyn til en eller flere attributter. I denne appen er det ikke brukt noe sortert, men er potensielt noe som ville kunne forbedret søk. Ved å higlighte atributten en sorterer, så ville en holdt orden på hvilke attributt(er) som er relevant ved søk.

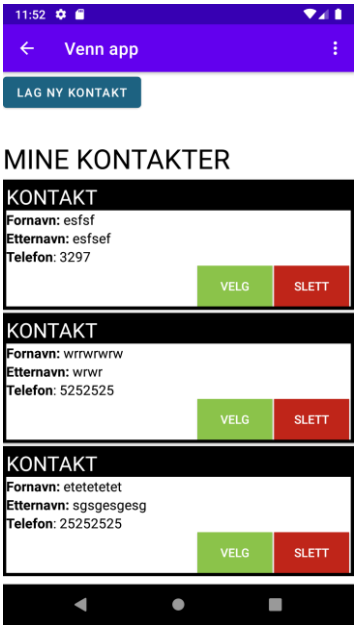
I appen som er laget, er det laget 2 lister; en for kontakter og en for avtaler. Begge viser all informasjon, pga ulike brukere potensielt kan være interessert i ulik form for informasjon. I kontakt registeret vil f.eks noen brukere kan søke med hensyn på telefonnummer for å finne navnet på en person, mens andre ønsker å finne telefonnummeret til en person gitt navnet på personen.

Kontakt liste

Designet i figur 1, tar utgangspunkt i en kombinasjon av kjennskap til bootstrap sin cardview [2] og at android også tilbyr cardviews[1]. Cards går ut på at en samler informasjon i en firkant, som gjerne er delt i hode-del og kropp-del. Hode-del fungerer da som en slags oversikt over det som er i kroppen, f.eks ved hjelp av et bilde og diverse informasjon som kan være viktig for å få overblikk over inneholdet. Cardview ser ut til å bruke gesalts lov om proximity [4], hvor en skaper visuell avstand mellom informasjon med å sepaerer dem i cards – som er atskilt fra hverandre.

I disse cards, er det bare brukt tittel. I selve kroppen er det brukt en visning over kolonne-verdi par assosiert med entries i tabellen kontakt. Kolonne navn skiller seg fra verdi, ved at det er brukt fet skrift.

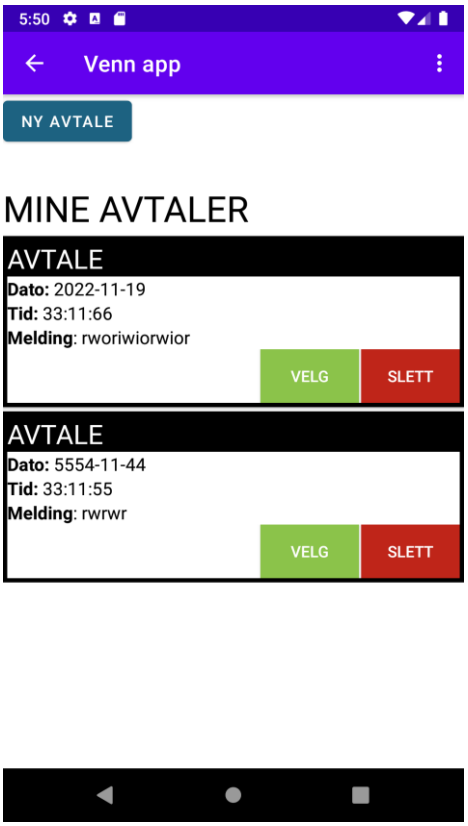
Figur 1: Liste over kontakter. Når en trykker velg, får man en mulighet til å endre kontakten. Det er også en knapp helt oppe, som en kan trykke på, som gir en mulighet til å lage en ny kontakt.



Avtale liste

Med samme argumenter på hvorfor det ble laget liste for kontakter, er det også laget lister over avtaler.

Figur 2: Liste over avtaler. Når en trykker velg, får man en mulighet til å endre avtalen. Det er også en knapp helt oppe, som en kan trykke på, som gir en mulighet til å lage en ny avtale.



Generelt om skjemaer for å registrere informasjon

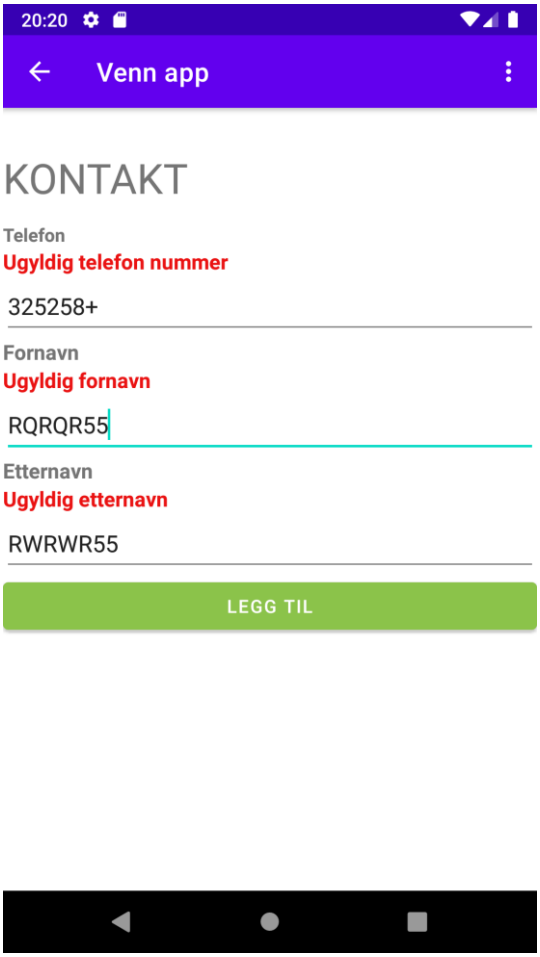
Et skjema er en måte å samle inn informasjon på. Da skriver en inn i felt, krysser av ting, og liknende. Når en er ferdig med dette, så sier en ifra at en er klar til å sende skjemaet.

Før skjemaet sendes inn, sjekkes skjemaet for åpenbare feil. Hvis det er feil i skjemaet, får brukeren informasjon om dette.

Lagre skjema for kontakt

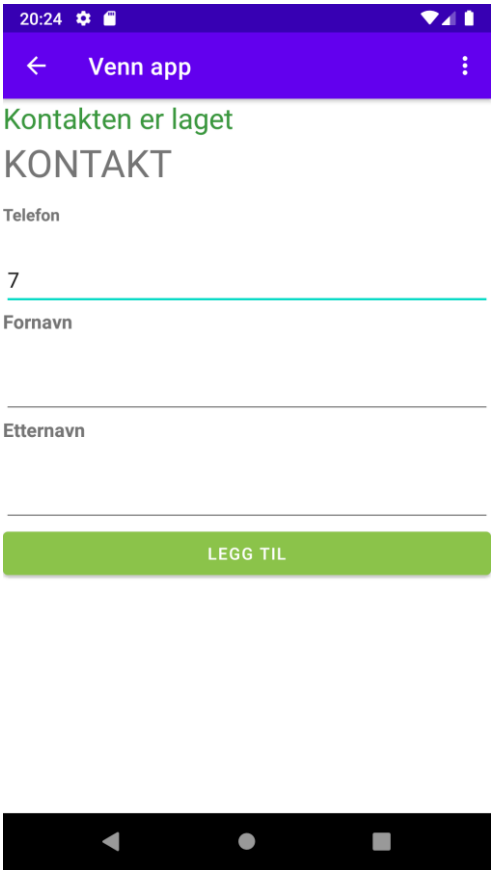
Det lagres informasjon om telefonnummer, fornavn og etternavn. Det er satt minlengde og makslengde på telefonnummer, fornavn og etternavn. Videre er det krav om at det kun er bokstaver i fornavn og etternavn, mens telefon kan ha + i begynnelsen (trenger ikke) etterfulgt av tall.

Figur 3



Når brukeren har laget en ny kontakt, får brukeren informasjon om at en ny kontakt er laget. Hvis brukeren etterpå registrer et nytt skjema med feiaktig input, fjernes informasjon om at kontakten er laget – kun vanlig feilmelding nær relevante input felt vises.

Figur 4



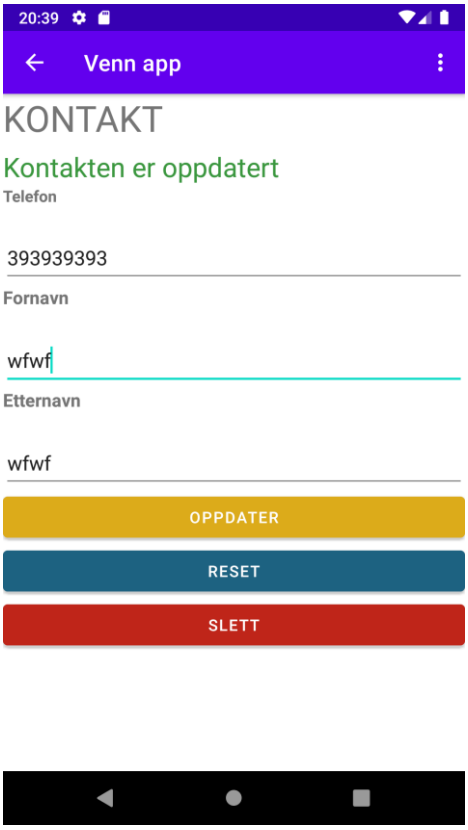
Endre skjema for kontakt

Skjemaet er det samme som for lagring. Forskjellen her er en hidden input felt som inneholder kontaktId for gjeldende kontakt. Videre så har en her mulighet til å oppdatere, resete skjema og å slette skjemaet.

Når en oppdaterer skjeamet, får en informasjon om at en har oppdatert. Når en trykker reset, så får input feltene tilbake verdiene tilsvarende de verdiene kontakt har. Når en oppdater, så vil reset forholde seg til de nye verdiene.

Hvis en trykker slett, vil kontakten slettes, samtidig som en hopper tilbake til listen over kontakter.

Figur 5

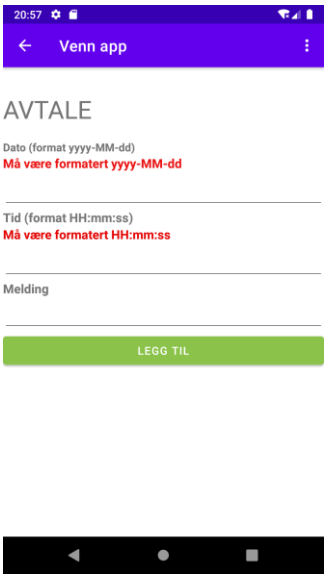


Lagre skjema for avtale

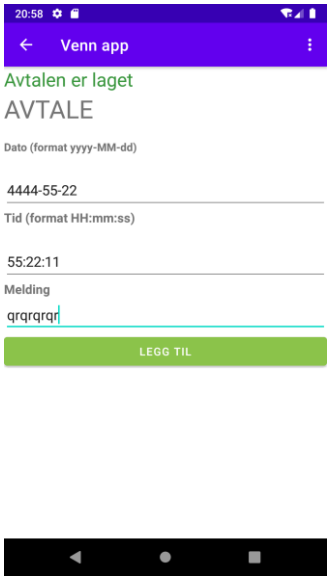
Avtale har kun input validering for dato og tid, ettersom det var en åpning i kravspesifikasjonen om at meldingen kunne være tom. En kunne hatt sjekk for makslengde, men er ikke gjort noe sjekk for dette her.

Dato og tid krever at brukeren skriver inn dato og tid på et format likt java.sql.Date og java.sql.Time. Ved å gjøre dette kan en sammenlikne strenger, ved at en prøver å forholde seg til kun et format av tid og dato. Årsaken til at en her bruke sql sin time og date, er pga den ikke forholder seg til Locale. Ved å sammenlikne datoer i databasen, med nåværende dato i dette formatet via java.sql.Date så har man en trygg ikke Locale avhengig måte å sammenlikne datoer på.

Figur 6



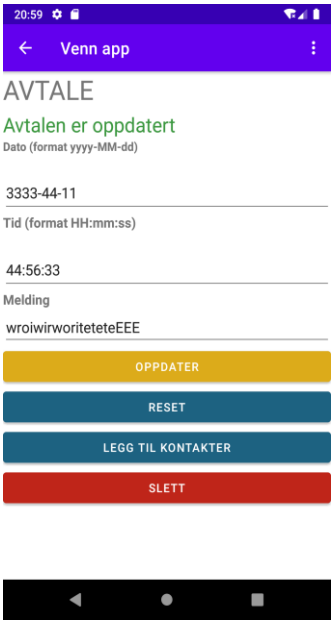
Figur 7



Endre skjema for avtale

Skjemaet for å endre avtale, er det samme skjemaet som brukes for å lage en avtale. Her har en mulighet til å endre, resete verdiene til verdiene avtalen opprinnelige hadde, legge til kontakter til avtale og å slette. Hvis en sletter, så vil en bli sendt tilbake til listen over avtaler.

Figur 8



Legge til kontakter til avtale

En kan legge til kontakter fra en avtale, og fjerne en kontakt fra en avtale.

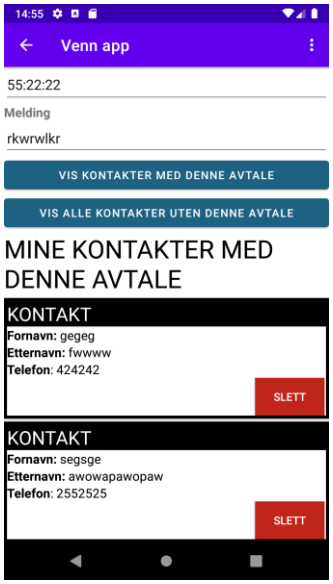
Når en legger til en avtale, så fjernes den far listen over kontakter en kan legge til. Samtidig legges denne kontakten til listen over kontakter som er assosiert med denne avtalen.

Hvis en går tilbake til avtalen, så sendes også informasjon om avtalen tilbake, slik at en fortsatt holder på med samme avtale.

Figur 9

Figur 10

Figur 11



Sende SMS på gitte tidspunkt og gi notifikasjon på gitte tidspunkt

På et et bestemt tidspunkt, hver dag, basert på et tidspunkt lagret i shared service, så vekkes en Alarm manger opp. Denne alarm manageren befinner seg servicen PeriodiskNofificationService. PeriodiskNotificationService aktiveres hver gang en trykker på en knapp for å starte SMS service i hovedmenyen. Da sendes en intent til en Broadcast receiver, som så starter servicen assosiert med intenten. Alarm manageren vil da prøve å aktivere to ulike services på et gitt tidspunkt hver dag. Hvis en trykker knapp assosiert med å stoppe SMS service i hovedmenyen, stoppes kun muligheten alarm manageren har med å aktivere SMS service.

Alarm manageren bruker inexact repating, fremfor exact, fordi en da kan spare strøm i følge dokumentasjoen som dukker opp når en trykker musen over den. Intervalet som er brukt er AlarmManager.INTERVAL_DAY ettersom den skal repeteres hver dag. Tidspunktet en først starter, er gitt i sharedprefrences. Når AlarmManager utfører sin scheduled oppgaver, så utfører den en Pending Intent som starter NotificationSendService og SMSSendService. Dissee PendingIntenten kan også nås via main menyen, og forsovidt resten av programmet, pga den er en pending intent av formen Pending.getService. Hvis begge utfører den samme Pending.getService opp mot samme

service ,vil begge sitte med samme PendingIntent(står i dokumentasjon når en tar musen over at den returnerer eksisterende eller ny pending intent). Dermed kan en, når en skal stoppe alarm manageren i å stoppe å gjøre en oppgaven, bare si at alarm manageren skal cancelle nøyaktig denne pending intent. Dvs at pending intenten, en fremtidig oppgave en kan gjøre, ikke blir utført.

NotificationSendService og SMSSendService utfører kun oppgavene sine en gang, for så å lukke ned, inntill den blir aktivert på nytt.

Sende SMS på gitte tidspunkt

I SMSSendService, så aktiveres aktiviteten SmsActivity gjennom å bruke en .send() gjennom en PendingIntent. Det er brukt PendingIntent, siden den da kjører mer uavhengig av resten av programmet. En Intent ville på en annen siden kjøre i samme kode.

Ettersom en starter en ny aktivitet, så vil nåværende aktivitet bli byttet ut med SMS aktiviteten. Brukeren blir da litt forstyrret når SMS sendes. Ettersom servicene kjører uavhengig av aktivitetene, så vil servicene fortsette å utføre det de skal.

Mye av koden av SMS lot seg ikke kjøre i service, men måtte gjøres i en aktivitet – slik at det ble nødvendig å ta SMS oppgavene i en egen aktivitet. SMSAcitivity bytter ikke ut layouten, men starter en ny aktivitet som fører brukeren ut i main menyen når SMS oppgavene er utført.

I SMS aktiviteten vil det sendes sms til alle kontakter som en har en avtale med denne dagen. I sms, vil de da dukke opp noe som likner figur 16. Noen ganger vil det da også dukke opp en sms beskjed som i figur 14.

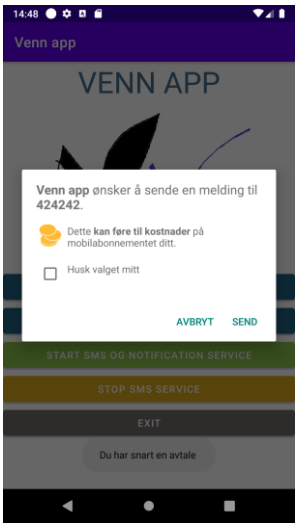
Sende notifikasjoner på gitte tidspunkt

Gjennom NotifacationSendService, sendes det 2 notifikasjoner, en gjennom en notifikasjons kanal slik at noe som likner figur 15 med «snart avtale» dukker opp. I selve applikasjonen vil noe som likner figur 14 dukke opp,dvs det gråe med tekst i.

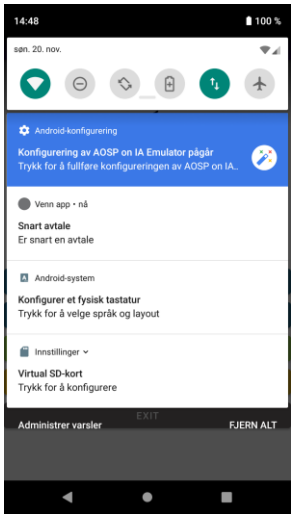
Det er notifikasjons servicen som står for å sende ut 2 ulike meldinger, en gjennom Toast.maketext , og en .notify() fra notificiation manager. Notifikasjon service gjør alt dette, nøyaktig 1 gang, når den blir aktivert av periodisk service 1 gang i døgnet på predefinert tidspunkt (satt i shared prefrence).

Notifikasjon servicen kan ikke deaktiveres, kun aktiveres. Sms service på en annen side, kan deaktiveres.

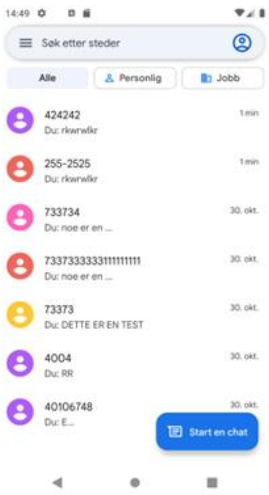
Figur 14



Figur 15



Figur 16



Content manager

Content manager er en mekanismer som muliggjør å dele data med andre applikasjoner, som muliggjør at tredjepart applikasjoner kan ta i bruk denne informasjonen via database spørringer i form av content:// [navn på provider]/[Entiet, som avtale og kontakt. Ofte omtalt som path]/id.

Content manager i appen, bygges omkring URI matcher som pattern matcher URI strenger. En har en eller flere path.En path en et slags abstrakt filter for å nå en submengde av data en øsnker å dele [5] Et eksempel på dette er /eple/form=sirkel/farge=rød/pris<30/#, hvor# da representerer id eller identifikator for en rad. Det er ikke gitt at id finnes, det betyr kun at en ser etter id'en etter en har tatt i bruk alle filtrene.

I URI matcher er det to kjente filtre, kalt # og *. # betyr tall og * betyr alt.

Fordi id bestående av flere kolonner ikke kan representerers av tall, så må en ta i bruk path for å begrense seg dataene til det en ønsker å modifisere/slette. F.eks kan en si kontakt/#/#, hvor en først bruker path til å begrense dataene med hensyn til første id, for så å behandle siste kolonnen som utgjør primærnøkkelen som en id.

Content providing av KontaktAvtale

Følgende filtre er brukt for å gi beskjed om en skal slette kun en rad, alle rader med en spesfikk avtale eller alle rader en spesifikk kontakt.

Hva en bruker, avhenger om en f.eks sletter en kontakt, en avtale eller en spesifikk kontaktavtale og dermed må slette noen av kontaktavtalene.

Kode 1

```
uriMatcher.addURI(PROVIDER, "kontaktavtale/#/#", KONTAKTAVTALESUB1);
uriMatcher.addURI(PROVIDER, "kontaktavtale/*/#", KONTAKTAVTALESUB2);
uriMatcher.addURI(PROVIDER, "kontaktavtale/#/*", KONTAKTAVTALESUB3);
uriMatcher.addURI(PROVIDER, "kontaktavtale/*/*", KONTAKTAVTALESUB4);
```

Videre er det sørget for at en kategoriserer disse riktig som type, dvs hvorvidt en kun har en eller flere path, eller om en også har id tilstede.

Kode 2

```
@Override
public String getType(Uri uri) {
    switch (uriMatcher.match(uri)) {
        case MKONTAKTAVTALE  | KONTAKTAVTALESUB2  | KONTAKTAVTALESUB3  |
KONTAKTAVTALESUB4:
            return "vnd.android.cursor.dir/vnd.example.kontaktavtale";
        case KONTAKTAVTALE | KONTAKTAVTALESUB1  :
            return "vnd.android.cursor.item/vnd.example.kontaktavtale";
        default:
            throw new
                    IllegalArgumentException("Ugyldig URI" + uri);
    }
}
```

Content providing av Avtale

Pga kun 1 kolonne for id, så er det kun brukt følgende pathes. En kan da enten slette en spesifikk avtalle eller alle avtalene.

Kode 3

```
uriMatcher.addURI(PROVIDER, "avtale", MAVTALE);
uriMatcher.addURI(PROVIDER, "avtale/#", AVTALE);
```

Content providing av Kontakt

Pga kun en kolonne for id, så er det kun brukt følgende pathes. En kan da enten slette en spesifikk kontakt eller alle kontaktene.

Kode 4

```
uriMatcher.addURI(PROVIDER, "kontakt", MKONTAKT);
uriMatcher.addURI(PROVIDER, "kontakt/#", KONTAKT);
```

Referanser

1. https://developer.android.com/develop/ui/views/layout/cardview, lest 30.10.2022
2. https://getbootstrap.com/docs/4.0/components/card/, lest 30.10.2022
3. Forelesningsnotater apputvikling 2022 vår oslomet, lest 30.10.2022
4. Yanxia Liang, **https://iopscience.iop.org/article/10.1088/1757-899X/392/6/062054/pdf,s.2**
5. https://developer.android.com/reference/android/content/ContentUris, lest 19.11.2022
6. http://personales.upv.es/thinkmind/dl/conferences/achi/achi_2018/achi_2018_4_10_20055.pdf, s.61-62 lest 19.11.2022

Appendix

SQL script for å lage databasen

.schema

```
CREATE TABLE android_metadata (Locale TEXT);

CREATE TABLE Kontakter(_ID INTEGER PRIMARY KEY, Fornavn TEXT, Etternavn TEXT, Telefon TEXT);

CREATE TABLE Avtaler(_ID INTEGER PRIMARY KEY, Dato TEXT, Tid TEXT,Melding TEXT);

CREATE TABLE KontaktAvtale (
        kontaktId       INTEGER,
        avtaleId        INTEGER,
        FOREIGN KEY(kontaktId) REFERENCES Kontakter(_ID),
        FOREIGN KEY(avtaleId) REFERENCES Avtaler(_ID),
        PRIMARY KEY(kontaktId,avtaleId)
);
```

Link til kode: https://github.com/s349967/vennApp