Mappe 2

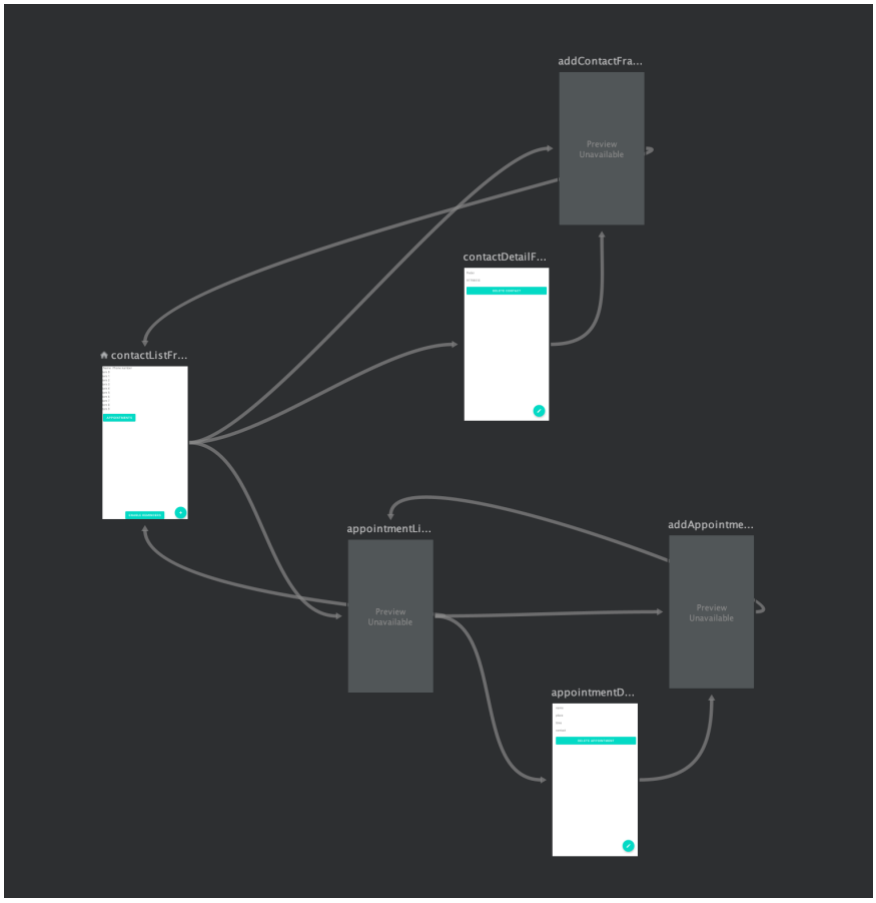
Apputvikling mappe 2 rapport

Peder Fosse - s354412
Rikard Dotzler - s313407

For å observere at SMS fungerer, sett tidspunktet for utsendelse til et tidspunkt nær deg. Dette gjøres i MainActivity. Utsendelse av SMS kan ta opp i mot 1 minutt etter satt tid

Navigering

Appen er bygd opp i fragmenter, sammen med navigation-component. Dette har vi lært oss å bruke ved hjelp av android developer sitt kurs, spesifikt i *Unit 3*. Ved bruk av fragmenter og navigation component får man tilgang til å lage en navigation_graph.xml som gir en visuell fremstilling av hvor brukeren kan bli sendt via de ulike fragmentene.



Via navigation component kan man også sende med argumenter til de ulike fragmentene, slik som for eksempel er gjort for å hente ut den riktige kontakten når detail_fragment blir startet, så detaljene til den riktige kontakten blir vist, og den riktige kontakten kan bli

oppdatert. Her er ett eksempel der navigation component blir brukt for å navigere fra ett fragment til et annet, mens vi sender med id'en til den rette kontakten med navigationsargumentene.

```
val adapter = ContactListAdapter { it: Contact
    val action = ContactListFragmentDirections.actionContactListFragmentToContactDetailFragment(it.id)
    this.findNavController().navigate(action)
}
```

Database

For å sette opp appen vår med database, har vi benyttet oss av Room. Vi har lært oss Room ved hjelp av blant annet android developers sitt kurs, mer spesifikt i *Unit 5.*

Først satt vi opp Entities i form av data klasser i kotlin filer:

```
8
9       @Entity(tableName = "appointment")
10      data class Appointment (
11          @PrimaryKey(autoGenerate = true)
12          val id: Int = 0,
13          val name: String,
14          val place: String,
15          val time: LocalDateTime,
16          val contactId: Int,
17      )
```

Deretter satt vi opp Data Access Objects (DAO), som Room oversetter til databasekall:

```
@Dao
interface AppointmentDao {
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insert(appointment: Appointment)

    @Update
    suspend fun update(appointment: Appointment)

    @Delete
    suspend fun delete(appointment: Appointment)

    @Query("SELECT * FROM appointment WHERE id = :id")
    fun getAppointment(id: Int): Flow<Appointment>

    @Query("SELECT * FROM appointment ORDER BY name ASC")
    fun getAppointments(): Flow<List<Appointment>>
}
```

I *ContactRoomDatabase.kt* satt vi opp selve databasen, her setter vi entities; altså hvilke tabeller som eksisterer, og også databaseversjonen, denne må oppdateres hver gang vi gjør

endringer i database schemaet. I tillegg har vi satt opp en typeconverter, slik at vi kan lagre datoer. Room vet nemlig ikke hvordan man kan lagre datoer, men hvis vi konverterer verdiene til Long kan vi lagre dem og konvertere hver gang vi legger inn / tar ut data fra databasen.

Til slutt har vi satt opp en viewModel for å abstrahere hele databasemodellen, denne viewModellen har også en factory i bunn som gjør at vi kan sette opp viewmodelen i fragmentene, som vist under:

```
class AddAppointmentFragment : Fragment() {
    private val navigationArgs: AddAppointmentFragmentArgs by navArgs()
    private var _binding: FragmentAddAppointmentBinding? = null
    private val binding get() = _binding!!

    private val viewModel: ContactViewModel by activityViewModels {
        ContactViewModelFactory(
            (activity?.application as ContactsApplication).database.contactDao(),
            (activity?.application as ContactsApplication).database.appointmentDao()
        )
    }
```

SMS og påminnelser

Ved oppstart av appen blir brukeren bedt om tillatelse til å sende SMS.
Brukeren har mulighet til å skru på påminnelser med SMS. Brukeren har så muligheten til å skru på Reminders med en toggle bryter. Når dette gjøres sendes et broadcast til en broadcastreceiver. Denne broadcast sin intent inneholder en extra boolean med navn “START”. Denne avgjør om broadcast receiveren starter eller stopper tjenesten. Dersom tjenesten skal startes starter servicen myPeriodic. Denne servicen har ansvar for å igangsette en alarm manager som gjentar et ønsket kall på en gitt intervall (24 timer) som starter fra tidspunktet satt i sharedpreferences. Når dette tidspunktet slår inn setter alarm manageren i gang en service kalt mySendService. Denne har ansvar for å sende ut SMS og notifikasjoner. Når denne servicen starter opprettes 2 observers. Den ene for live data fra getAppointments() og den andre for live data fra getContacts(). Myperiodic starter mysendservice på oncreate for å sette i gang observere. Deretter vil alarm manageren vente 10 sekunder før den utfører utsendelsen for å sikre at dataene er hentet. Deretter startes funksjonen send Messages. Denne funksjonen filtrerer avtalene ned til de som er i dag, henter ut informasjon fra avtalen og sender det som SMS til kontakten som er knyttet opp med avtalen. Dersom det er minst 1 avtale i dag vil dette bli sendt som notifikasjon til telefonen gjennom notifikasjons kanalen opprettet i mainactivity. Deretter hentes default message fra shared preference som legges inn i meldingen dersom avtalen ikke har noen melding knyttet til seg.

Brukerflyt

Brukerflyten består hovedsakelig av 2 aktiviteter. 1 Kontakter, og 2 Avtaler. Dette er intuitivt da appens funksjonaliteter begår rundt kontakter og avtaler med kontaktene. For hvert skjermbilde er det en tydelig rund “+” knapp der man intuitivt kan legge til en ny kontakt eller avtale avhengig av hvilken side man er på. Kontaktene og avtalene listes også for brukeren som gjør at man alltid vet hvilket innhold man har lagt inn. Videre kan man trykke seg inn på hver individuelle kontakt eller avtale og endre dem eller slette dem. Dette oppfyller full CRUD (Create, read, update, delete) på dataene. Før man sletter data vil man bli spurt av et popup vindu om bekreftelse for å unngå feilaktig sletting. Ettersom SMS reminders er en så sentral del av appens funksjonalitet har det blitt plassert slik at det alltid er tilgjengelig for brukeren.

Istedenfor å bruke listview, har vi benyttet oss av RecyclerView fordi det ble anbefalt av android dokumentasjonen (https://developer.android.com/reference/android/widget/ListView). Her står det i dokumentasjonen av recyclerview er en mer moderne, fleksibel og *performant* måte å vise data i lister på.

Designvalg

Appens visuelle uttrykk tilnærmer seg Android sitt naturlige og standard uttrykk. Fargevalget avviker fra det standard lilla til en utilitaristisk turkis som gir en mer rolig atmosfære. Videre faller appen inn under et helhetlig utilitaristisk uttrykk der kun det du trenger er det som vises til brukeren. Appen har store åpne områder, både for å lede brukerens øyne til det som er viktig, og for å ha plass til å vise mye data på samme skjermbilde når brukeren fyller appen med data.