ITSMAP-Theme Rapport
Pacemaker Data

Gruppemedlemmer:
Aksel Brandt, #10319
Frank Laumann Nielsen, #20113466
Mads Kring, #201271195

Afleveret d.13/10-2015

Indhold

Ansvarsfordeling:

Frank:

- Bound Service = dataservice
- SQLite = SqlConnect
- Model = PacemakerDataObject

Mads:

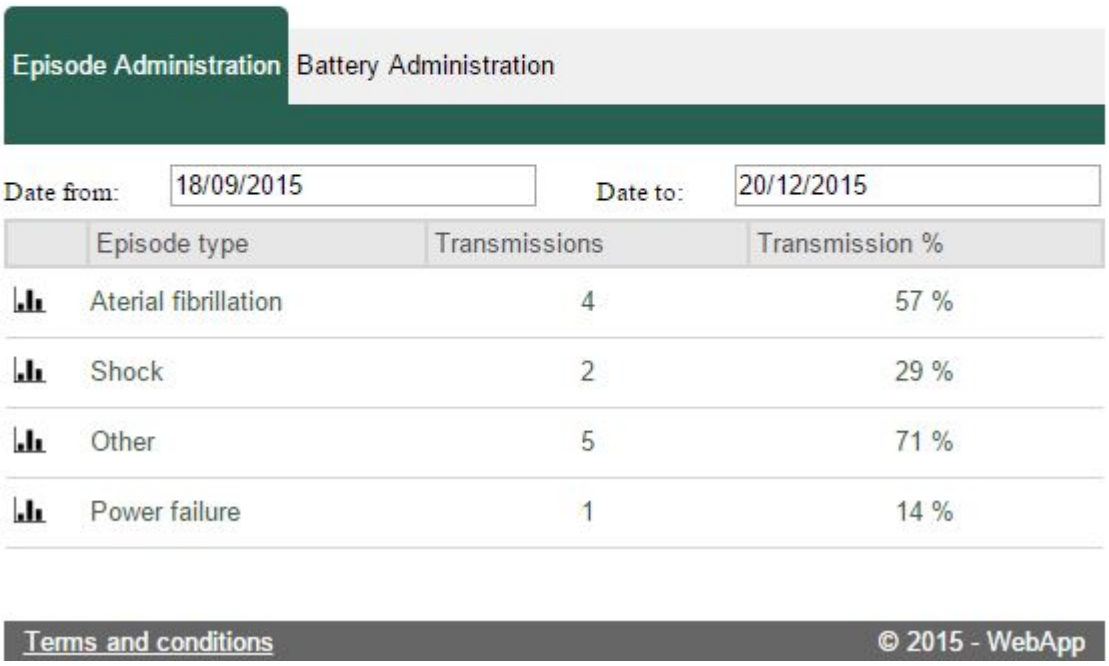
- Activity = EpisodeStatisticDetails
- Fragment = PictureFragment
- Intentservice = TimerIntentService

Aksel:

- Layout = .xml
- Activity = EpisodeStatistic

1 Indledning

En pacemaker gør livet for folk med ustabil hjerterytme lidt mere sikker. Pacemakeren går ind og sender elektriske impulser til hjertet så det holder en fast rytme. Det kan derfor være en fordel for sundhedspersonalet at undersøge hvad pacemakeren har foretaget sig. For at kunne tilgå det på en nem måde ønskes en app udviklet, der kan vise hvilke typer af Episoder der er forekommet og hvornår.



Figur 1: Billede af Web app.

Ovenfor ses et screendump af den web app, der i forvejen er udviklet, hvor man nemt kan se hvilke episoder der oftest opleves. Vi vil lave en tilsvarende oversigt, som mobil app, over hvilke episoder, der generelt oftest opleves for alle pacemaker patienter, hvilket kan give sygehuspersonalet en bedre ide om hvad der kan forventes flest haste konsultationer. Altså vi vil for hver episode type angive hvor mange sendinger den optræder i og dermed hvor mange procent af alle sendinger.

2 Krav

Her beskrives krav for projektet. Kravene vil blive uddybet i use cases, som vil give indblik i den funktionalitet app'en skal have for at kravene er opfyldt. For at kunne forstå nogle af koncepterne gives her en forklaring af termerne episode og sending.
En sending er en besked der sendes fra en pacemaker. En sending kan indeholde en eller flere episoder, endda flere episoder af samme type.
Eksempel:
Sendingen S er modtaget fra en pacemaker med episoderne A, B, C, D og E.
Sendingen T er modtaget fra en pacemaker med episoderne F og G.
Episoderne A, C og F er af typen atrieflimmer. Episoden B er af typen shock. Resten af episoderne er forskellige.
I dette eksempel er der 2 sendinger. Episode Typen atrieflimmer findes i begge sendinger, så episode typen atrieflimmer får antal sendinger sat til 2. Episode Typen shock findes kun i den ene sending så den får antal sendinger til 1. Det samme er gældende i procent, episode typen atrieflimmer findes i 2 ud af 2 sendinger, dermed i alle de eksisterende sendinger, altså 100 %.

2.1. Use Case 1 - Tilgå episodestatistik

Denne Use Case er den grundlæggende funktionalitet. Det skal være muligt at kunne se antal sendinger hver episode type har indgået i, samt den procentvise del af alle sendinger.

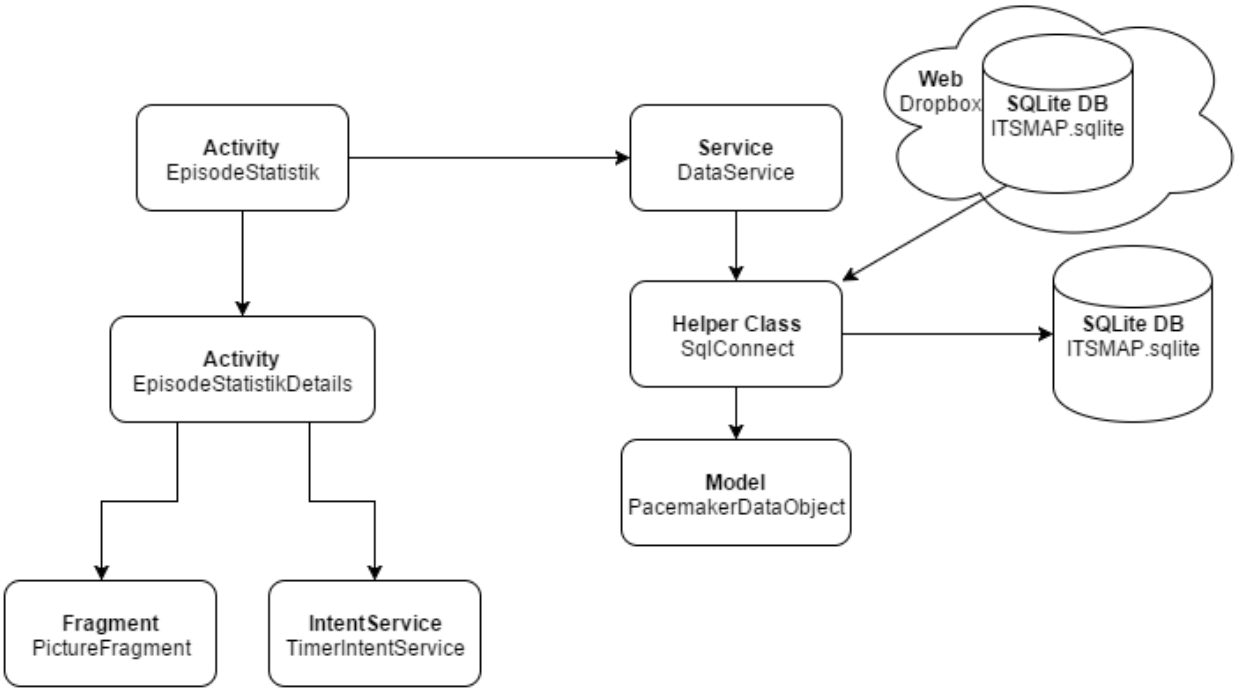
2.2. Use Case 2 - Tilgå detaljer

Denne Use Case er en udvidelse af Use Case 1. Da det ikke er muligt at vise alle informationer på en mobil device der ikke er ligger ned, kan man trykke på en episodetype for at se de ikke viste detaljer.

2.3. Use Case 3 - Tilgå episodestatistik diagram

Denne Use Case beskriver den grafiske visning, af dataen fra en pacemaker. Diagrammet skal give et overblik over hver episode types statistik. Diagrammet skal vise en statistik over antal episoder pr måned over det sidste år.

3. Komponent model



Figur 2: Komponent model for projektet.

Komponentmodellen herover giver et samlet overblik over hvordan mobil app'en er blevet implementeret i dette projekt. Modellen kan læses som følgende. Der startes i EpisodeStatistik Activity'en. Dette er det første skærmbillede der bliver vist når app'en åbnes. Pilene viser hvilke komponenter, der herfra er tilgængelige. F.eks. vil det være muligt at skifte skærmbilledet til EpisodeStatistikDetails.
Modellen viser også hvordan der benyttes en lokal SQLite database, der kan opdateres ved at hente en opdateret version og erstatte den lokale database med denne.

4 App lifecycle af projektets activities

4.1 onCreate

I denne metode initialiseres de forskellige views, der findes på hver activity, i globale variabler. Det er en fordel at gøre dette her, fordi det behøves kun at blive gjort en gang i app'ens lifecycle. Dermed skal disse views ikke initialiseres efter at app'en har været minimeret, som hvis der var valgt at lægge initialiseringen af views i metoden onResume.
Desuden bindes servicen DataService i metoden onCreate. Dette er valgt af samme begrundelse, at det er ønskværdigt at servicen fortsat kører i baggrunden når app'en er minimeret.

4.2 onStart og onStop

Er ikke brugt i dette projekt. Hvis der, i modsætningen til dette projekt, var brug for at køre en eller flere funktioner når app'en starter op igen efter at have være minimeret ville onStart være nødvendig.

4.3 onResume

I dette projekt er der valgt at lægge registreringen af broadcastreceiver i metoden onResume. Dette er valgt da afregistreringen af broadcastreceiveren er valgt placeret i onPause.
Desuden opdateres databasen i onResume.

4.4 onPause

I metoden onPause afregistreres broadcastreceiveren. Når app'en er paused er der ingen grund til at lytte efter broadcasts.

4.5 onDestroy

Her nedlægges activity'en. Dette betyder at bindingen af servicen ophæves.

5 “Finurligheder” i koden

Her vil der blive skrevet om nogle kodeafsnit fra projekt.

5.1 Downloading af databasefil

Da der ikke var mulighed for at få hostet en online database, blev der valgt at hente en databasefil, som kunne bruges som en lokal database.

```
89      // Inspired from: http://www.helloandroid.com/tutorials/how-download-fileimage-url-your-device
90      public void getDbFile()
91      {
92          try{
93              URL url = new URL("https://www.dropbox.com/s/h4lom5ara4fd9h3/ITSMAP.sqlite?dl=1");
94              String outFileName = SqlConnect.path +"/"+ SqlConnect.dbName;
95
96              URLConnection con = url.openConnection();
97              InputStream is = con.getInputStream();
98
99              // Open the empty db as the output stream
100             OutputStream myOutput = new FileOutputStream(outFileName);
101
102             // transfer bytes from the inputfile to the outputfile
103             byte[] buffer = new byte[1024];
104             int length;
105             while ((length = is.read(buffer)) != -1)
106             {
107                 Log.d("Download", "" + length);
108                 myOutput.write(buffer, 0, length);
109             }
110             // Close the streams
111             myOutput.flush();
112             myOutput.close();
113             is.close();
```

Figur 3: Kode for funktionen getDbFile.

Her ses koden for download af ny en database fil. Når der hentes en fil fra nettet modtages den i bytes. De modtagne bytes gemmes til den lokale fil via en OutputStream i mindre dele indtil der ikke er flere bytes at læse. Når alle bytes er læst lukkes OutputStream og InputStream.

5.2 PacemakerDataObject til lagring af database entries

Data fra databasen kan lagres i en masse variabler, der ikke hænger sammen og som derved let kan blive rodet sammen uhensigtsmæssigt. Data’en kunne også lagres i et objekt der er tilpasset netop denne data og dens struktur. Det er hvad der er valgt i dette projekt.

```
8   public class PacemakerDataObject {
9       private String episodeType_;
10      private int transmissionsId_;
11      private String date_;
12
13      public PacemakerDataObject()
14      {
15      }
16      public PacemakerDataObject(String episodeType, int transmissionsId, String date)
17      {
18          episodeType_ = episodeType;
19          transmissionsId_ = transmissionsId;
20          date_ = date;
21      }
22      public String getEpisodeType() { return episodeType_; }
26      public void setEpisodeType(String episodeType) { episodeType_ = episodeType; }
30      public int getTransmissionsId() { return transmissionsId_; }
34      public void setTransmissionsId(int transmissionsId) { transmissionsId_ = transmissionsId; }
38      public String getDate() { return date_; }
42      public void setDate(String date) { date_ = date; }
46      public static ArrayList<PacemakerDataObject> getList()
47      {
48          ArrayList<PacemakerDataObject> result = new ArrayList<~>();
49
50          result.add(new PacemakerDataObject("Aterial Fibrillation", 1, "20151220113452"));
51          result.add(new PacemakerDataObject("Shock", 1, "20150926103452"));
52          result.add(new PacemakerDataObject("Aterial Fibrillation", 1, "20151220113352"));
53          result.add(new PacemakerDataObject("Other", 2, "20150927213452"));
54
55          return result;
56      }
```

Figur 4: Klassen PacemakerDataObject til lagring af data fra databasen.

Klassen er en model over det pacemaker data der ligger i vores database. Klassen indeholder tre variabler: episodeType_ der er en string, transmissionId_ der er et tal og date_ der er en string. For hver variabel er der en funktion til at sætte en værdi på variablen og en funktion til at få fat i den nuværende værdi. Nederst i koden ses en statisk metode, der er blevet brugt til test i starten, før databasen var oppe og køre.
Med denne klasse har det været enklere at holde styr på hvilke variabler der hørte til hvilken entry. Det er samtidig nemt at proppe en masse af disse objekter ind i et array også løbe igennem alle objekter og udskrive, sammenligne eller gemme en ny værdi til de forskellige objekter. Klassen gør det også oplagt at validere på de værdier der forsøges gemmes i de forskellige objekters variabler. Validering er dog ikke benyttet i dette projekt.

5.3 Episode type optælling og procentregning

Dette kode afsnit optæller episode typer i de forskellige sendinger. Selve koden for dette afsnit er meget langt, derfor er der kun taget de første par linjer med i udsnittet fra koden herunder. Resten af koden kan ses i filen “EpisodeStatistic.java” under funktion “broadcastReceived” i projektet.

```
99      // Method called by onReceived broadcast
100     private void broadcastReceived()
101     {
102         // Get all entries from db
103         list = mService.getData();
104
105         // Make a list of all episodeType
106         ep = new ArrayList<String>();
107         for (int i = 0; i < list.size(); i++) {
108             if (!ep.contains(list.get(i).getEpisodeType())) {
109                 ep.add(list.get(i).getEpisodeType());
110             }
111         }
112         //...
```

Figur 5: Første linjer af funktionen braodcastReceived.

I koden ovenfor hentes data fra servicen og gemmes i variablen list. Andet trin er at lave en liste der indeholder alle de forskellige episode typer en og kun en gang.
Det følgende kode findes ikke i denne rapport, men kan som før nævnt findes i den egentlige kode, der er afleveret sammen med denne rapport. Den resterende kode i denne funktion vil fortsat blive gennemgået her.
Efter listen med unikke episode typer er oprettet, optælles der, for hver episode type, hvor mange sendinger den fremkommer i. Dernæst optælles der hvor mange sendinger der i alt er hentet fra databasen, som benyttes til at beregne den procentvise del af sendinger denne episode type forekommer i. Til sidst oprettes ArrayAdapters for at kunne fordele det udledte data i de forskellige ListView's.

5.4 Valg af Landscape eller Portrait mode.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("test", "" + getResources().getConfiguration().orientation);
    if (getResources().getConfiguration().orientation == 1)
    setContentView(R.layout.activity_episode_statistic_port);
     else if (getResources().getConfiguration().orientation == 2)
        setContentView(R.layout.activity_episode_statistic_land);
       else
        setContentView(R.layout.activity_episode_statistic_port);
```

Figur 6: Orientation funktionen

EpisodeStatistic activity har en væsentligt anderledes adfærd ved forskellig orientation, da der vises 3 listviews i landscape mode i modsætningen til kun ét i portrait mode.
getResources().getConfiguration().orientation giver orienteringen på telefonen. Er den 1 er telefonen i portrait mode og 2 er den i landscape mode.
Der laves derfor en if-else sætning som starter activity_episode_statistic_port.xml op hvis deviceet er i portrate mode og åbner activity_episode_statistic_land.xml. Hvis den hverken giver 1 eller 2 startes activity_episode_statistic_port.xml som standard.

5.5 Data mellem activities

Dataen som EpisodeStatisticDetails activityen bruger, kommer fra EpisodeStatistic activityen, via en intent som ses nedenfor:

```
Intent intent = getIntent();
episodeType = intent.getStringExtra("episodeType");
amount = intent.getIntExtra("transmissions", 0);
percentage = intent.getIntExtra("procentTransmission", 0);
PacemakerDates = intent.getStringArrayListExtra("dates");
```

Figur 7:

Denne data bruges til at bestemme hvad der skal vises i søjlediagrammet og fragmentet der kan startes fra EpisodeStatisticDetails activity.

5.6 Søjlediagramoptegnelse

Til at optegne et søjlediagram over episodernes forekomst, er et library der hedder mpandroidchartlibrary-2-1-4 anvendt. Her vises den primære kode ifm. dette:

```
BarData data = new BarData(labels, dataset);
chart = new BarChart(getBaseContext());
chart.setData(data);
chart.setDescription("");
```

Figur 8:

Dataen fra databasen er klargjort og omstruktureret til variablerne “labels” og “dataset” som mpandroidchartlibrary bruger til at optegne søjlediagrammet.

5.7 TimerIntentService

I EpisodeStatisticDetails onCreate() startes en IntentService der sender en broadcast hver 45. sekund for at meddele hver gang en person får implantet en pacemaker på verdensplan. Et kodeudsnit for dette ses her:

```
try {
    Thread.sleep(45000);
} catch (InterruptedException e) {
    e.printStackTrace();
}

Intent i = new Intent("secondPassed");
LocalBroadcastManager.getInstance(this).sendBroadcast(i);
```

Figur 9

Det smarte ved en IntentService er at den automatisk kører i en tråd for sig selv, så sleep-funktionen ovenfor ikke blokerer user interaction. IntentServicen bruger en LocalBroadcastManager til at opdatere EpisodeStatisticDetails. Som det ses på billedet sendes der ikke nogle extras med på Intenten i broadcast, da den eneste funktion der ønskes er broadcasten i sig selv.

5.8 onSaveInstanceState(Bunde state)

For at gemme en variabel der skal beholde dens værdi når en activity stopper og dens onCreate() kaldes igen bruges state som ses herunder.

```
public void onSaveInstanceState(Bundle state) {
    super.onSaveInstanceState(state);
    state.putInt(KEY_VALUE,PMCounter);
```

Figur 10

Dette gør vi for at undgå at miste dens værdi når der gåes til landscape mode da onDestroy() nedlægger vores activity.

5.9 PictureFragment

I ActivityEpisodeDetails startes et fragment der viser billeder alt efter hvilken episodetype der er valgt. Her ses et kodeudsnit af implementeringen af dette:

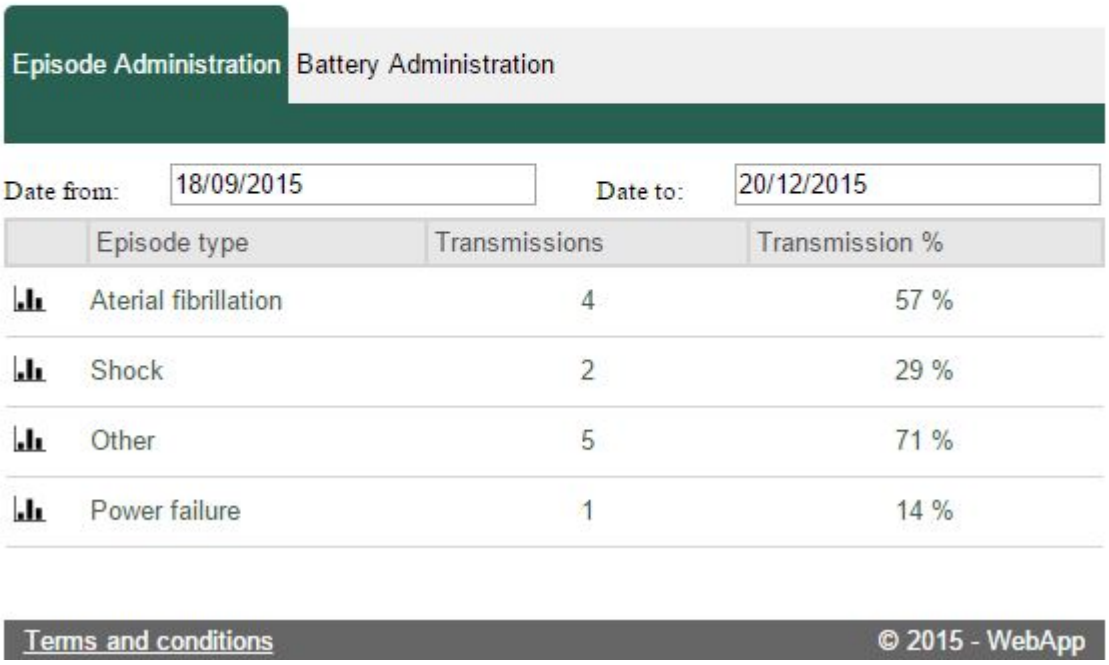
```
FragmentTransaction transaction = manager.beginTransaction();
transaction.add(R.id.statistic_details_layout, frag,"picfrag");
transaction.commit();
```

Figur 11

Som det ses, bruges "manager" som er en FragmentManager til at starte fragmentet. Fragments er gode at bruge, hvis man kun ønsker at ændre en lille del af skærmen, og fragments kan kalde metoder fra activityen. Vi har også et fragment med for at demonstrere brugen af det.Fremtidig udvikling
Her vil der blive diskuteret hvilket fremtidigt arbejde appen kunne blive bedre af.

6 Layout

Det var fra starten meningen at efterligne layoutet fra Franks bachelor-webapp:



Figur 12: Billede af web app.

Det er ikke blevet implementeret pga. tidspres. Det er ønskværdigt at layoutet er forholdsvist ens på forskellige platforme, da det højner brugervenligheden. Det er dog ikke målet at få det kopieret ned til mindste detalje, da der er forskel på layout mellem mobil apps og web apps. Det der skulle være kopieret er bl.a. farver, ikoner og skrifttyper, som noget af det basale.

6.1 Vælg tidsrum og andre ekstra funktionaliteter

I web app projektet findes der yderligere use cases end dem der er præsenteret i dette projekt. Bl.a. er det muligt i web app'en at vælge et tidsrum at se data for. Desuden skal web app'en videreudvikles så man har mulighed for at fravælge nogle episode typer man ikke er interesseret i. En anden funktionalitet, som web app'en får, er at man kan vælge og fravælge patienter, så der også er mulighed for at se denne data for en eller flere bestemte patienter. Den sidste funktionalitet, der er valgt at beskrive her, er en oversigt over nuværende implanterede pacemakere og et estimat på hvornår de skal skiftes, der løbende bliver beregnet. Dette er en funktionalitet personalet, der skal udskifte pacemakeren, kan bruge til at effektivisere processen.

6.2 Vis batteristatus på pacemakeren.

En anden mulig videreudvikling kunne være en batteristatus activity. Det vil give sundhedspersonalet mulighed for at vurdere hvor lang tid der er til at patienten skal indkaldes til udskiftning af batteriet. Telefonen skal connectes med pacemakeren via bluetooth. På den måde kan spændingen på pacemakerens batteri sende til telefonen og et ca tal for batteriets strømniveau findes.

7 Konklusion

App'en der er udviklet gennem projektet virker og har de vigtigste funktioner, som er beskrevet i synopsen. De ønskede data fremvises til brugeren som ønsket. Der er forskel på hvor meget data der vises alt efter om telefonen er i portrait mode eller landscape mode, og der vælges layout med hensyn til devicets resolution. De to activities varetager henholdsvis overordnet visning af de forskellige episodetyper og detaljeret information vedrørende den valgte episodetype. Der bruges en SqlLite server til dataen. Denne kan opdateres via nettet. Dette varetages af en service. I EpisodeStatisticDetails bruges et fragment til at vise billeder. En service opdaterer tallet for hvor mange pacemakere der er blevet implanteret.
Databasefilen kan opdateres eksternt, og app'en bruger den nyeste data. Alt tekst der bliver vist i app'en er externalized og app'en kan dermed nemt blive oversat ved at udfylde en specialisering af string.xml og lægge den i en ny value mappe, med det nye sprogs forkortelses bogstaver. F.eks. er der i dette projekt blevet oprettet mappen values-da for en dansk version af app'en.
Der er flere ting vi gerne ville have gjort mere ud af, bl.a. designet af appens layout, et par flere funktioner og generelt mere finesse i app'en.
En ting der desværre ikke blev tid til var at få designet til at matche webapplikationen.