

### 0.0.1 Komponent 1: Mobil applikation

Denne komponent er hovedformålet med hele systemet, og har til formål at formidle alt bus information til brugerne. Den gør det muligt for brugeren at se busruter indtegnet på et kort, med stoppesteder og positionen for de busser der kører på ruten. Ydermere vil det være muligt at se hvor lang tid der er, til den næsten bus ankommer til et valgt busstoppested.

#### Design:

Mobil applikationen er blevet udviklet til android mobiltelefoner der kører med OS version 4.3 Jelly Bean og op til 4.4.2 KitKat. Dertil er der blevet udviklet imod android API level 18 og 19.

Mobil applikationen er udviklet med fokus på, at der skulle ske så lidt arbejde på telefonen som muligt, det skal i stedet for ske alle tunge udregninger og dataprocessering op en webservice *Se afsnit 8.2.2 Komponent 2: Mobile service*

For at kunne vise busruter og stoppesteder, busser og tid til ankomst, skal dette hentes fra en databasen. Til dette er der blevet lavet 2 klasser, TrackABusProvider og SoapProvider. TrackABusProvideren er udviklet som en bound service, som både BuslistMenuActivity og BusMapActivity bruger. For at bind til TrackABusProvider, bliver startService() kaldt, dette vil starte services hvis den ikke allerede kører, efter det vil bindService blive kaldt, dette binder, i dette tilfælde, BuslistMenuActivity til TrackABusProvideren.

#### Kodeudsnit 1: Binding til TrackABusProvider

```
1 if (ConnectivityChecker.hasInternet) {  
2     Intent intent = new Intent(BuslistMenuActivity.this, ←  
        TrackABusProvider.class);  
3     startService(intent);  
4     bindService(intent, Connection, Context.BIND_AUTO_CREATE);  
5 }
```

Denne binding vil ske asynkron, og derfor ikke til at sige hvornår den er færdig, for at se hvornår den er færdig, bruges en ServiceConnection. Denne service connection har en callback function, onServiceConnected(), der vil blive kaldt så snart den er færdig med at bind. For at sikre at de kald der kræver at der er blevet bundet til services skal derfor

ligges her i. `binder.getService()` vil initialisere en instance af den service der bliver bundet til, og fra den være muligt at kalde de forskellige service funktioner der er implementeret i `TrackABusProvider` klassen.

#### Kodeudsnit 2: `ServiceConnection` i `BuslistMenuActivity`

```
1 private ServiceConnection Connection = new ServiceConnection() {
2     @Override
3     public void onServiceConnected(ComponentName name, IBinder ←
4         service) {
5         LocalBinder binder = (LocalBinder ) service;
6         BusProvider = binder.getService();
7         mBound = true;
8         UpdateArrayOfBusses();
9     }
10    @Override
11    public void onServiceDisconnected(ComponentName name) {
12        mBound = false;
13    }
14 };
```

Alle funktionerne i `TrackABusprovider` klassen bliver afviklet i deres egen tråd for ikke at blokere `main/UI` tråden. For at `TrackABusProvider` funktionerne kan sende data rigtig tilbage til den klasse der kalder funktionen, bliver der brugt en `Message Handler`.

På `TrackABusProvider` siden er dette blevet lavet ved at hver funktion tager imod en `int ReplyMessage` og en `Handler ReplyTo` som parameter. Når funktionen er færdig med at få data fra `SoapProvideren`, *Se afsnit 9.2.1 Implementering af persistens i mobilapplikationen for information om hvordan MySQL databasen bliver tilgået*, bliver der oprettet en ny `Message`. `ReplyTo` handleren bliver brugt så beskeden ved hvor den skal hen, og `ReplyMessage` er så den kan fortælle den handler den bliver sendt til, hvad den er blevet færdig med. I kodeudsnit 3 kan der ses et eksempel på en funktion i `TrackABusProvider` klassen. `GetbusRoute()` tilgår `soapProvider` klassen, der ligger i data tilgangs laget, for at hente en bestemt busrute, og alle de stoppesteder der ligger på ruten. Data hentet fra `MySQL` databasen bliver derefter lagt i et `ParcelableArrayList`, dette gør det muligt at sende custom datatyper med i de messages der bliver sendt. Til sidst bliver der oprettet en ny `Message`, med en `ReplyMessage`, så den handler der modtager beskeden ved hvad der er

blevet modtaget, den nye Message får så sat data på sig, og bliver sendt afsted til replyTo handleren, der er implementeret i en af de klasser der er bundet til TrackABusProvideren.

### Kodeudsnit 3: GetBusRoute() i TrackABusProvider

```
1 public void GetBusRoute(final String busNumber, final int ↵
    ReplyMessage, final Handler replyTo){
2 try{
3     new Thread(new Runnable() {
4         public void run() {
5             mMessenger = new Messenger(replyTo);
6             Bundle b = new Bundle();
7             ArrayList<BusRoute> arg0 = soapProvider.GetBusRoute(↵
                busNumber);
8             ArrayList<BusStop> arg1 = soapProvider.GetBusStops(↵
                busNumber);
9             b.putParcelableArrayList("BusRoute", arg0);
10            b.putParcelableArrayList("BusStop", arg1);
11
12            Message bMsg = Message.obtain(null, ReplyMessage, 0, 0);
13            bMsg.setData(b);
14            try {
15                mMessenger.send(bMsg);
16            } catch (RemoteException e) {
17                e.printStackTrace();
18            }
19        }}).start();
20 }
```

De klasser der er bundet til TrackABusProvideren skal implementere en handler, der skal bruges for at kunne kalde funktionerne i TrackABusProvideren, denne handler står for at modtage beskederne, når TrackABusProvider funktionen er færdig. For at kalde en af TrackABusProvider funktionerne skal der, som beskrevet ovenstående, altid sendes en *int ReplyMessage* og en *Handler ReplyTo*. ReplyTo parameteren er den handler der skal modtage beskederne, kodeafsnit 4 viser message handleren i BusMapActivity. Når den modtager en besked, vil den tjekke på hvilken ReplyMessage der ligger i beskeden, for at håndtere beskeden rigtigt. For at få det data der bliver sendt med i beskeden, bliver der kaldt en `msg.getData()` på beskeden som vist i kodeafsnit 4.

#### Kodeudsnit 4: msgHandler i BusMapActivity

```
1 final static public int BUS_ROUTE_DONE = 1;
2 final static public int BUS_POS_DONE = 2;
3 ...
4 class msgHandler extends Handler{
5 @Override
6     public void handleMessage(Message msg) {
7         if(msg != null){
8             switch(msg.what){
9                 case BUS_ROUTE_DONE:
10                     ...
11                     ArrayList<BusRoute> BusRoutes = msg.getData().←
                        getParcelableArrayList("BusRoute");
12                     ArrayList<BusStop> BusStops = msg.getData().←
                        getParcelableArrayList("BusStop");
13                     ...
14                     break;
15                 case BUS_POS_DONE:
16                     ...
17                     break;
```

For at kunne indtegne ruterne, stoppesteder og busserne skal der bruges et kort at tegne på, hertil er der blevet brugt Google maps. For at kunne bruge dette map, kræves en API key fra google, der fås fra googles API console. API key'en skal derefter skrives ind i android manifest filen, dette gør det muligt at bruge et google map i sin applikation.

#### Kodeudsnit 5: API key i Manifest

```
1 <meta-data
2     android:name="com.google.android.maps.v2.API_KEY"
3     android:value="AIzaSyC9qLxvm9yVIBJ5Dp0VqMapFvc4VLU1qu8"/>
```

For at vise selve kortet, skal der laves en layout fil, der indeholder et mapFragment, som vist i kodeudsnit ???. Det vil nu være muligt at se et kort i applikationen.

#### Kodeudsnit 6: mapFragment

```
1 <fragment
2   android:id="@+id/map"
3   android:layout_width="match_parent"
4   android:layout_height="wrap_content"
5   android:name="com.google.android.gms.maps.MapFragment" />
```

Det er nu muligt at indtegne ruter, stoppesteder og busser på ruten. Det bliver gjort efter at have modtaget ruten fra TrackABusProvider klassen. Ruten bliver tegnet ved brug af en Polyline, der bliver tegnet ved at bruge de GPS-koordinater der er blevet hentet fra MySQL databasen.

#### Kodeudsnit 7: hvordan en Polyline bliver indtegnet på kortet

```
1 PolylineOptions pOption = new PolylineOptions().width(10).color(0x66ff0000);
2 for(int i = 0; i < points.size(); i++){
3   pOption.add(points.get(i).Position);
4 }
5 map.addPolyline(pOption);
```

Busstoppestederne bliver tegnet ind ved at bruge de GPS-koordinater der er blevet hentet fra databasen, til at oprette nye markers på kortet. Der vil blive knyttet en ClickListener på alle markerne, der vil begynde at hente tiden, til den næste bus ankommer ved det valgte stoppested og opdatere skærmen så brugeren kan se det.

#### Kodeudsnit 8: Hvordan stoppesteder bliver indtegnet på kortet

```
1 for(int i = 0; i < stops.size(); i++){
2   map.addMarker(new MarkerOptions()
3     .position(stops.get(i).Position.Position).title(stops.get(i).Name)
4     .icon(BitmapDescriptorFactory.fromResource(R.drawable.teststop)));
5 }
```

Der er blevet implementeret mulighed for at favorisere busruter, dette gøres fra mainmenu\_layout view'et. for at gøre dette nemt og pænt for brugeren har det været nødvendigt at lave en custom listadapter, dette før det muligt at lave en liste, hvor hver list element

både består af navnet på rute en ToggleButton og en ProgressBar

Mange steder i applikationen skal der bruges adgang til internet, og der kan derfor forkomme fejl når dette sker. Dette er blevet håndteret ved at implementere en BroadcastReceiver i klassen ConnectivityChecker, der abonnerer på, der sker ændringer i netværksforbindelsen for telefonen, som set i kodeafsnit ???. Det vil nu være muligt at tjekke på om der er internet forbindelse før man prøver at kalde funktioner der kræver at der er forbindelse til internettet.

#### Kodeudsnit 9: ConnectivityChecker i manifest filen

```
1      <receiver android:name="ConnectivityChecker">
2          <intent-filter>
3              <action android:name="android.net.conn.↵
4                  CONNECTIVITY_CHANGE"/>
5              <action android:name="android.net.conn.↵
6                  WIFI_STATE_CHANGED"/>
7          </intent-filter>
8      </receiver>
```