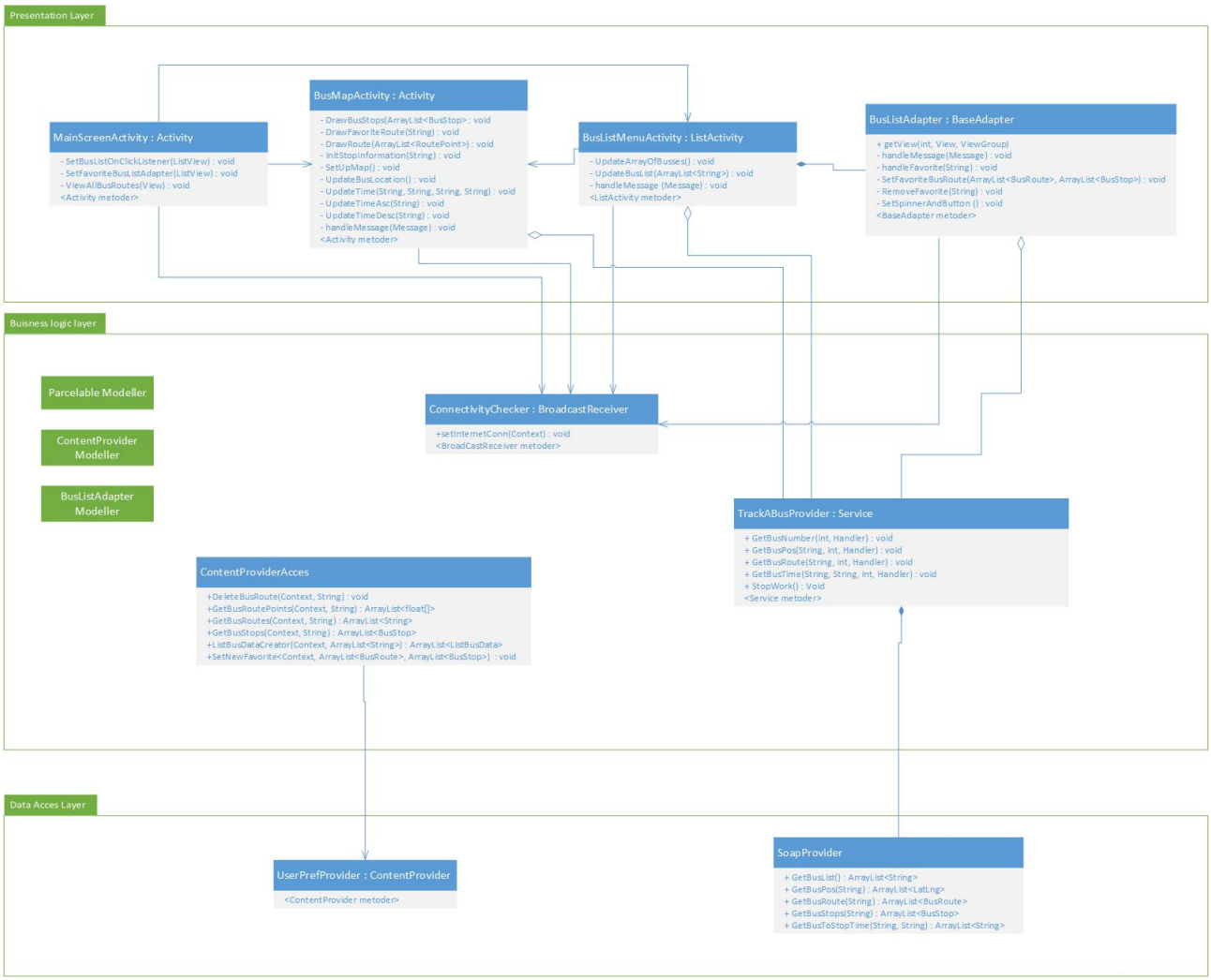
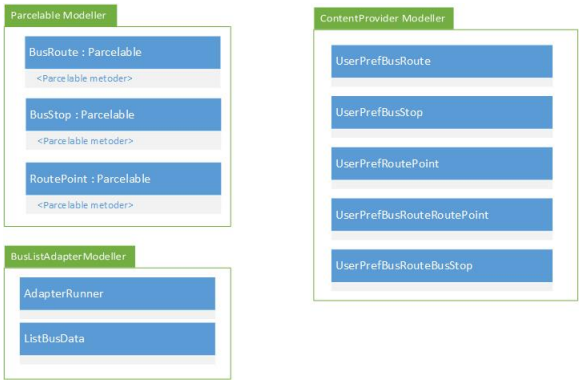
0.0.1 Komponent 1: Mobil applikation

Denne komponent har til formå at formidle alt bus information til brugerne. Den gør det muligt for brugeren at se busruter med stoppesteder indtegnet på et kort samt positionen for de busser der køre på ruten. Herudover vil det være muligt at se hvor lang tid der er til, at den næsten bus ankommer ved et valgt busstoppested. På figur 1 kan de forskellige klasser ses, samt hvilke lag de ligger i og hvordan de interagerer. Model klasser vises ikke som interageringer.



Figur 1: Klassediagram for mobil applikationen

På figur 2 ses de forskellige model klasser, der hovedsageligt består af custom datatyper.



Figur 2: Klassediagram for model klasser i mobil applikationen

Design:

Applikationen er blevet udviklet til android mobiltelefoner som kører med OS version 4.3 Jelly Bean og op til 4.4 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. Alle tunge udregninger og processeringer sker på en webservice, *Se afsnit 8.2.2 Komponent 2: Mobile service*

For at kunne vise busruter, stoppesteder, busser og tid til ankomst, skal dette hentes fra en databasen. Til dette formål er der blevet lavet to klasser, TrackABusProvider og SoapProvider. TrackABusProvideren er udviklet som en BoundService, som både BuslistMenuActivity og BusMapActivity bruger. For at binde til TrackABusProvideren, bliver "startService"kaldt, som starter servicen, hvis den ikke allerede kører. Herefter vil funktionen "bindService"blive kaldt, som i dette tilfælde binder 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 asynkront, og det er derfor ikke muligt at vide, hvornår den er færdig. Til dette formål bruges en ServiceConnection. Denne har en callback function, "onServiceConnected", der vil blive kaldt så snart der er blevet bundet til servicen. For at sikre at de funktioner, som kræver en bundet service, først bliver kaldt når dette er opfyldt, bliver kaldt i onServiceConnected. "getService"vil initialisere en instans af den service der er bliver bundet til, og herfra vil det 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 ↩
       service) {
4     LocalBinder  binder = (LocalBinder ) service;
5     BusProvider = binder.getService();
6     mBound = true;
7     UpdateArrayOfBusses();
8   }
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 rigtigt tilbage til den klasse der kaldte funktionen, bliver der brugt en Message Handler.
På TrackABusProvider siden er dette blevet implementeret ved, at hver funktion tager imod en ReplyMessage og Handler som parameter. Når funktionen er færdig med at hente data fra SoapProvideren, bliver der oprettet en ny Message. Se *afsnit 9.2.1 Implementering af persistens i mobilapplikationen* for information om, hvordan MySQL databasen bliver tilgået. Messagen sendes over den medfølgende Handler, som er implementeret i den klasse, der kaldte servicen. ReplyMessage beskriver overfor Handleren, hvad den er blevet færdig med, så Handleren kan udføre det korrekte arbejde med dataen. På kodeudsnit 3 kan der ses et eksempel på en funktion i TrackABusProvider klassen. "GetbusRoute"tilgår

SoapProvider klassen, for at hente en bestemt busrute, og alle de stoppesteder der hører til denne fra MySQL databasen. Data hentet herfra, bliver lagt i en ParcelableArrayList. Dette gør det muligt at sende custom datatyper med i de messages, der bliver sendt. Disse custom datatyper skal dog implementere Parcelable interfacet. Disse klasser kan ses på 2, under "Parcelable Modeller". Til sidst bliver der oprettet en ny Message, med den ReplyMessage der fulgte med som parameter, så Handleren ved hvilket arbejde den skal udføre. Denne message sendes til Handler parameteren.

Kodeudsnit 3: GetBusRoute() i TrackABusProvdider

```
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 Message Handler, der skal bruges i sammenhæng med kald til funktionerne i TrackABusProvideren. Denne Handler står for at modtage beskederne, når den kaldte TrackABusProvider funktion er færdig. For at kalde en af TrackABusProvider funktionerne skal der, som beskrevet ovenstående, altid medsendes en ReplyMessage og en Message Handler. Handler parameteren er den Message Handler der er oprettet i den klasse som kalder servicen. På kodeudsnit **??** vises Message Handleren i BusMapActivity. Når den modtager en besked, vil den undersøge hvilken ReplyMessage der ligger i beskeden. Dette gøres for at være sikker på, at data håndteret korrekt. For at få det data der bliver sendt med i beskeden, bliver "getData"kaldt på beskeden.

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, stoppestederne og busserne skal det være muligt at have et kort at tegne på. Hertil er der blevet brugt Google Maps.[1] For at kunne bruge dette kort, kræves det at en API nøgle fra Google bliver tilføjet til manifestet. Denne kan anskaffes fra Googles API konsol[2]. 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="AIzaSyC9qLxvm9yVIBJ5Dp0VqMapFvc4VLUlqu8"/>
```

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.

[1] Dette kan der læses mere om på https://developers.google.com/maps/documentation/android/
[2] Denne tilgås igennem https://code.google.com/apis/console

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 efer 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(0↩
    x66ff0000);
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 abonnere på, der sker ændringer i netværksforbindelsen for telefonen, som set i kodeafsnid **??**. 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.↩
                      CONNECTIVITY_CHANGE"/>
4                 <action android:name="android.net.conn.↩
                      WIFI_STATE_CHANGED"/>
5             </intent-filter>
6         </receiver>
```