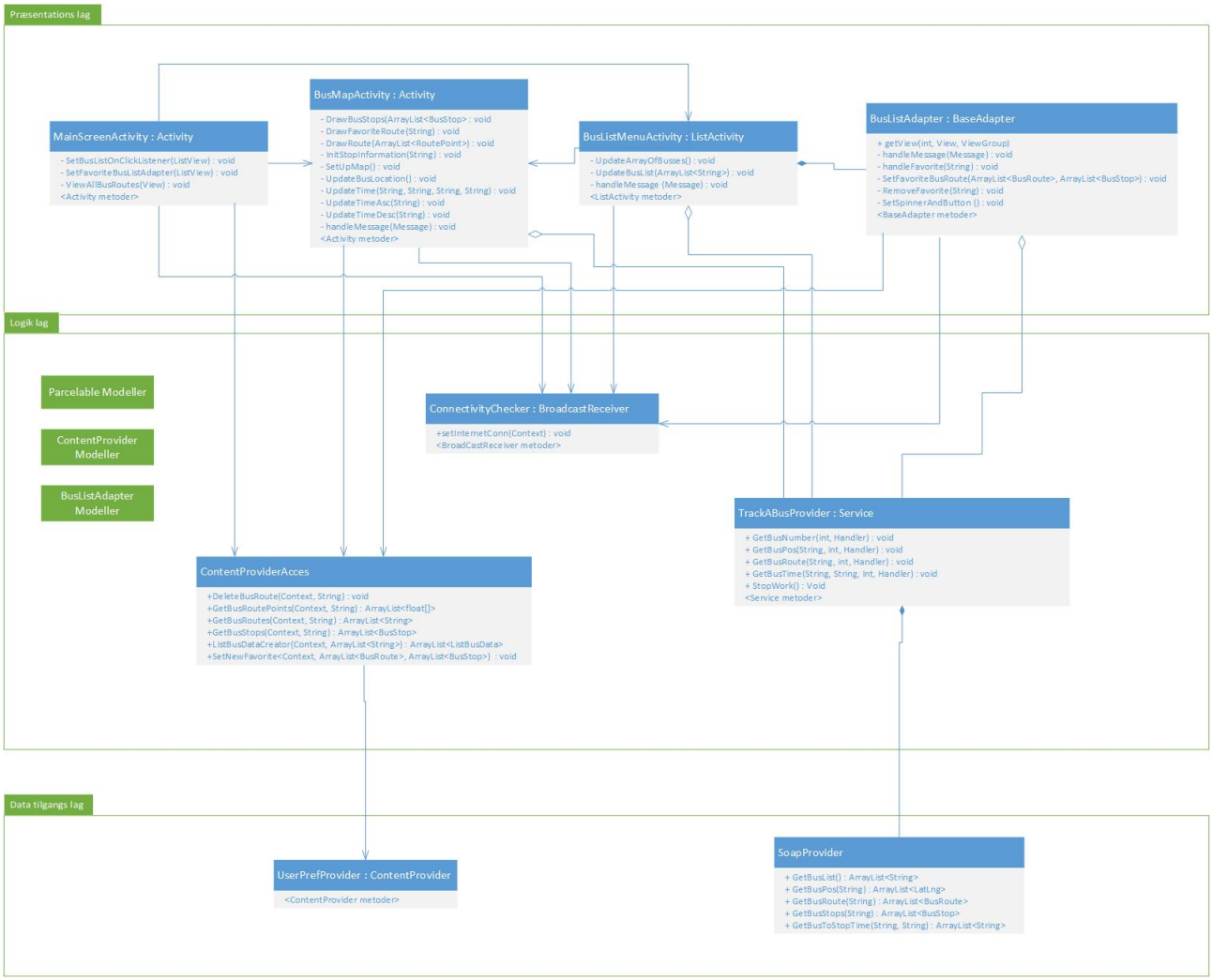
0.0.1 Komponent 1: Mobil applikation

Denne komponent har til formål at formidle alt bus information til brugeren. 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.

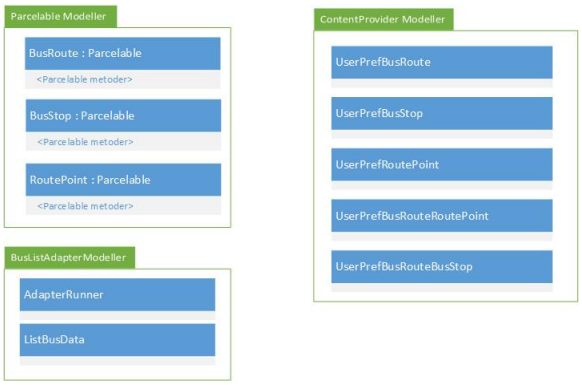
Specifikationer

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

Her følger en kort beskrivelse af hver klasse, samt den funktionalitet klassen tilføjer til systemet.

- MainScreenActivity
  - Denne klasse implementerer Activity interfacet, hvilket betyder, at den bruges som et view. Denne agerer som startskærm for applikationen, og lader brugeren vælge en favorit bus. Herfra kan brugeren desuden også vælge at åbne BusListMenuActivity.
- BusListMenuActivity
  - Denne klasse implementerer ListActivity interfacet, hvilket betyder, at den bruges som et view, men en list adapter kan sættes hertil. Den har til formål at præsentere brugeren for listen af ruter der eksisterer i MySQL databasen, samt favorisere en given rute. Selve viewet er bygget op med en adapter, hvori favoriserings funktionaliteten ligger, samt click eventet for favoriserings knappen. Selve klikket på et element fra listen, håndteres i BusListMenuActivity klassen. Herigennem har brugeren mulighed for at vælge en busrute, favoriseret såvel som ikke.
- BusMapActivity
  - Denne klasse implementerer Activity interfacet, hvilket betyder, at den bruges som et view. Den har til formål at præsentere brugeren for et kort med

indtegnet valgt busrute og stoppesteder. Her vil kørende busser på ruten også præsenteres. Den har desuden til formål at starte tidsopdaterings funktionalitet, ved et tryk på et stoppested.

- BusListAdapter
  - Denne klasse implementerer Adapter interfacet, hvilket betyder, at den kan bruges til at repræsentere de layout-elementer der skal vises på viewet. I denne sammenhæng er den koblet til BusListMenuActivityet. Den har til formål at håndtere alle favoriserings funktionerne, og click eventet for favoriserings knappen.
- ConnectivityChecker
  - Denne klasse implementerer BroadcastReceiver, hvilket betyder, at den modtager registrede system events. Disse events bliver sat i manifestet. I denne sammenhæng er den koblet til CONNECTIVITY_CHANGE og WIFI_STATE_CHANGED. Disse events omhandler ændringer i netværksforbindelse. Når der sker en ændring, undersøger den hvorvidt nettet kan tilgås, hvorefter den sætter en statisk bool til true hvis det kan tilgås, og false hvis ikke.
- ContentProviderAcces
  - Denne klasse er et abstraktions lag mellem præsentations laget og ContentProvideren, som ligger i data tilgangs laget. Dette betyder at det er den eneste klasse, som tilgår ContentProvideren. Funktionerne i denne klasse er statiske, da kun én indskrivnings- eller sletnings process kan udføres af gangen.
- TrackABusProvider
  - Denne klasse implementer Service interfacet, hvilket betyder at klassen kan tilgås som en Bound Service. Klassen agerer som et abstraktions lag mellem præsentations laget og SoapProvideren, som ligger i data tilgangs laget. Dette betyder, at den er den eneste klasse, som tilgår SoapProvideren. Funktionerne i denne klasse returnerer aldrig direte til det view som kaldte den, men i

stedet over en MessageHandler. Dette gør at service funktionerne kan afvikles asynkront.

- UserPrefProvider
  - Denne klasse implementer ContentProvider interfacet, hvilket betyder at klassen kan tilgås igennem "getContentResolver"funktionen, hvis den er koblet til applikationen i manifestet. Denne klasse sørger for at tilgå den lokale SQLite database, for at persistere eller fjerne en favoriseret rute. Denne er således den ene klasse i data tilgangs laget. Der kommunikeres udelukkende med denne klasse igennem ContentProviderAcces klassen.
- SoapProvider
  - Denne klasse har til formål at tilgå de forskellige funktioner i mobil servicen på serveren. Disse kald tilgår MySQL databasen, og er derfor den anden klasse i data tilgangs laget. Der kommunikeres udelukkende med denne klasse igennem TrackABusProvideren.
- Parcelable modeller
  - Disse klasser bruges som datatyper TrackABusProvideren. Disse har alle til fælles at de implementerer Parcelable. Dette gør det muligt for disse modeller at pakkes ned i en message, og sendes til en nessage handler.
- ContentProvider modeller
  - Disse klasser er modeller for de tabeller, SQLite databasen indeholder. De indeholder herunder navnene på de forskellige kolonner i tabellen, samt hvilken URI, der skal kaldes med, for at tilgå den givne tabel.
- BusListAdapter modeller
  - Disse klasser bruges i sammenhæng med at gemme de UI elementer, som bruges i BusListAdapteren, samt det data denne indeholder.

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 4 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].

Kodeudsnit 5: API nøgle 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, som indeholder et mapFragment, som vist i kodeudsnit 6. 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, på dette fragment. Rute og bus relvant information hentes igennen TrackBusProvideren, og på kodeudsnit 7 kan det ses, hvordan en rute tegnes. Ruten bliver tegnet ved brug af en Polyline, som opbygges af de koordinater der hentes 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 ligeså tegnet ind ved at bruge de GPS-koordinater, der er blevet hentet fra MySQL databasen. Disse koordinater bruges til at tegne markører på kortet. Der vil blive knyttet en ClickListener på alle markerne, som kaldes når et stoppested trykkes. Denne sørger for at starte tidsopdaterings funktionerne. På kodeudsnit 8 kan det ses, hvordan et stoppested tilføjes til kortet.

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 }
```

BusListAdapteren er lavet til det formål, således at hvert list element i BusListMenu-Activity, kan indeholde en togglebutton, en progressbar, foruden tekst. Dette gøres da

favoriserings metoder også startes fra en af disse elementer.

Mange steder i applikationen skal der bruges adgang til internettet, og der kan derfor forkomme fejl, hvis der ikke er tilgang hertil. Dette er blevet håndteret ved at implementere en BroadcastReceiver i klassen ConnectivityChecker, der abonnerer på, ændringer i netværksforbindelsen for både wifi og mobilt data netværk. På kodeudsnit 9, kan manifest filen ses, hvori det vises hvordan BroadcastReceiveren kobles til de nødvendige events. Når der sker et event undersøges der, om der er internet, og en statisk bool sættes til true hvis der er forbindelse, og false hvis der ikke er. Hertil er det muligt for applikationen til hver en tid at vide, om der er adgang 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>
```