Android Wear Workshop

Christoffer Marcussen
og
Sindre Nordbø

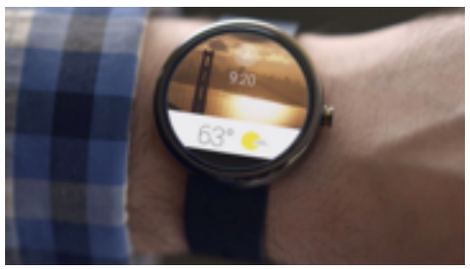
http://bekk.github.io/android-wear-kurs/
https://github.com/bekk/android-wear-kurs

Agenda

- Intro til Android Wear
- Basic komponenter og utvikling
- Kommunikasjon
- Notifications
- Lister
- Oppgaver

- Skal ha introworkshop til Android Wear
- Fokus på koding, lite teori. Det ligger mer teori i Git-wikiene som dere kan bruke under oppgaveløsning.
- Still spørsmål underveis!
- Denne workshopen er work in progress. Er lagt inn som en Labs her i BEKK, så alle er velkomne til å bidra.

Android Wear



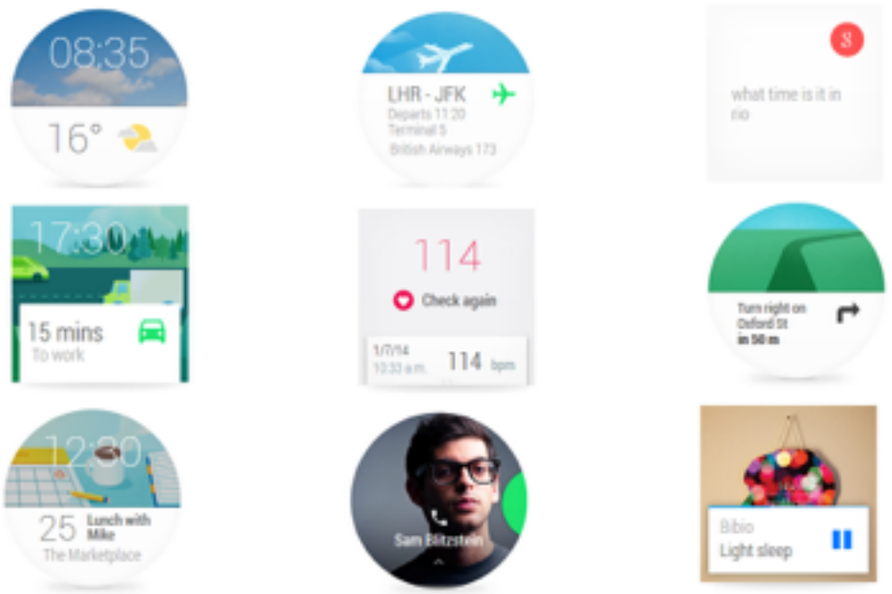
- Klokker for Android. Store produsenter er Samsung, LG, Motorola. Kommer: Tag Heuer, Huawei
- Developer preview lansert tidlig i 2014. Klokker lansert på I/O 2014.

Android Wear

- Gjøre informasjon tilgjengelig når du trenger det
- Tilkobling over Bluetooth
  - Companion app

- Informasjon vises når brukeren trenger det. Feks Google Now kort som viser hvor lang tid det tar å kjøre hjem fra jobb
- Noen klokker har wifi-støtte. Andre bruker Wifi gjennom telefonen
- Android Wear er en klar forlengelse av mobilen, der Notifications står sentralt. En Wear-app er en del av appen på telefonen. Oppkoblingen skjer via en såkalt companion app på telefonen over Bluetooth

Android Wear



- Værnotifications
- Notifications som er lignende det man får på telefonen for feks tekstmeldinger
- Spotifyavspilling
- Samtaler

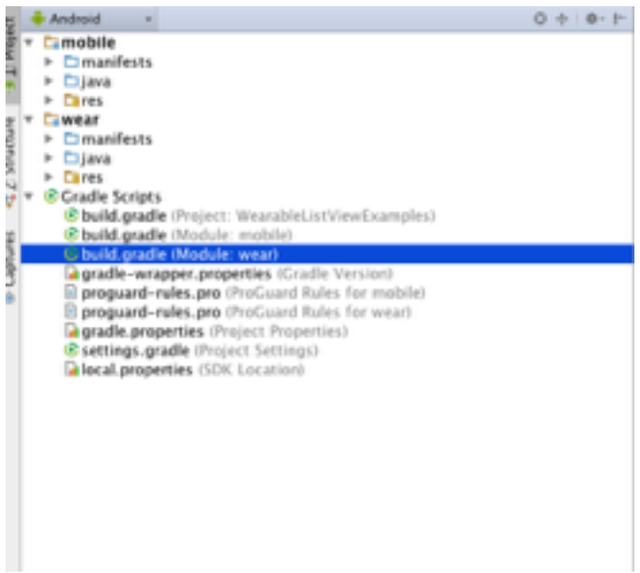
Gjør generelt at man ikke trenger å se på mobilen dersom det skjer noe

Utvikling

- Android Studio + SDK
- Telefon/emulator
- Klokke/emulator

- Bruker Android Studio for utvikling. Dersom man er vant til å kode Android er det basically business as usual
- For utvikling trenger man enheter eller emulatorer. Litt innviklet å koble emulator mot emulator, så anbefaler å ha fysisk Android-telefon. Må ha enhet som har minimun Android 4.3

Moduler



- Setter opp moduler i Android Studio, en for mobil og en for klokke. Disse har separate Gradle-filer og begge registreres i settings.gradle. Alt av kildefiler er separat. Kan dersom man vil legge inn i en felles Library module som begge depender på.
- Man kan også legge til moduler for feks TV, Auto eller Glass

Layout

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.wearable.view.WatchViewStub
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools" android:id="@+id/watch_view_stub"
    android:layout_width="match_parent" android:layout_height="match_parent"
    app:rectLayout="@layout/rect_activity_main" app:roundLayout="@layout/round_activity_main"
    tools:context=".MainActivity"
    tools:deviceIds="wear"></android.support.wearable.view.WatchViewStub>
```

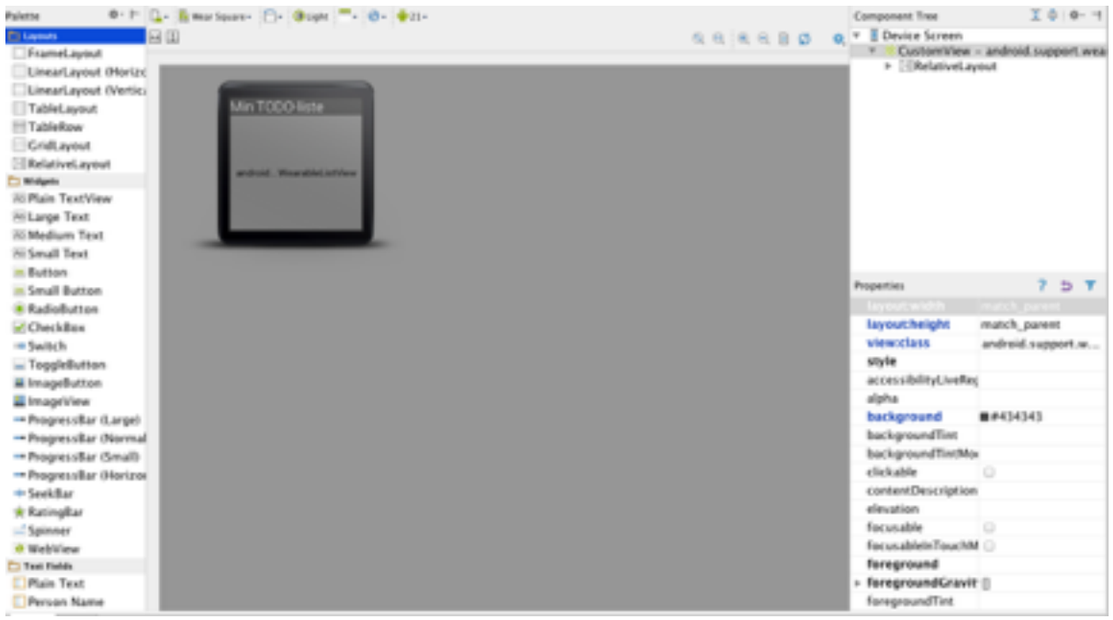
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" tools:context=".MainActivity" tools:deviceIds="wear_round">

    <TextView android:id="@+id/text" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:layout_centerHorizontal="true"
        android:layout_centerVertical="true" android:text="@string/hello_round" />
</RelativeLayout>
```

- Har for Android Wear utvikling to layoutfiler: En for firkantede skjermer og en for runde. Har så en hovedfil der disse to inkluderes. I denne filen vil det runtime velges ut hvilken av de to som brukes, basert på tilgjengelig skjerm.

  Dette er litt den samme tankegangen som når vi utvikler for telefoner og kan bruke forskjellige layoutfiler for forskjellige skjermstørrelser
- Øverste bildet viser hovedfilen, nederste viser eksempel på rund layout

Layout



- Bruker samme XML-editoren som før, med drag-and-drop interface

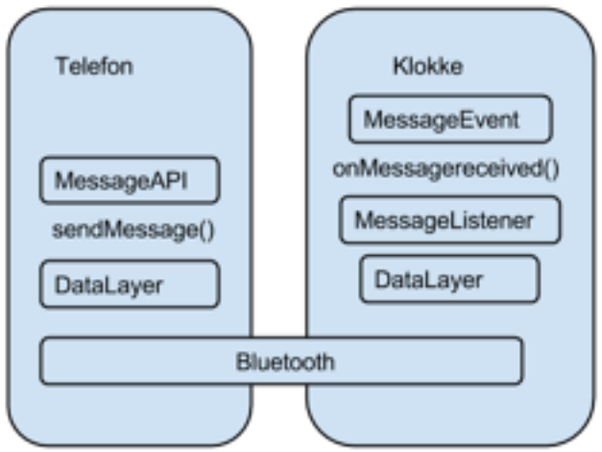
Kommunikasjon

- Message API
  - Enveiskommunikasjon. Eks. knappetrykk på telefon viser ei melding på telefon
- Data Items
  - Synkronisering av data. Eks. vedlikehold av ei liste på både telefon og klokke
- Assets
  - Sending av større data som bilder, via blobs

- For å kommunisere mellom telefon og klokke har vi flere muligheter. Felles for alle er at de bruker Google Play Services, som gir oss tilgang til Googles API'er, samt Bluetooth
- Message API kan brukes til Remote Procedure CAlls
- Data Items brukes ved sync av data opp mot 100k.
- Data kan også legges til som en Asset. Her kan man sende større mengder data som bilder
- Skal fokusere på Message API og Data Items

Message API

- Melding består av en path og en payload



- En melding kan sendes til flere tilkoblede enheter. Det betyr at dersom jeg sender en melding som skal starte en activity, så må jeg være sikker på at noden faktisk har denne og kan starte den.
- Først må vi koble opp en GoogleApiClient
- Dereter bruker vi MessageAPI til å sende en melding

Message API
- avsender

```
GoogleApiClient mGoogleApiClient = new GoogleApiClient.Builder(this)
        .addConnectionCallbacks(new ConnectionCallbacks() {
            // Implementasjon av onConnected og onConnectionSuspended
        })
        .addOnConnectionFailedListener(new OnConnectionFailedListener() {
            // Implementasjon av onConnectionFailed
        })
        .addApi(Wearable.API)
        .build();


private void sendMessage(final String path, final String content) {
      Wearable.MessageApi.sendMessage(mGoogleApiClient, nodeId, UNIQUE_PATH, new byte[0])
              .setResultCallback(new ResultCallback<MessageApi.SendMessageResult>() {
                  @Override
                  public void onResult(MessageApi.SendMessageResult sendMessageResult) {
                      // Kall på sendMessageResult.getStatus().isSuccess() for å se status på meldingen
                  }
                  );
              }
}
```

- En melding kan sendes til flere tilkoblede enheter. Det betyr at dersom jeg sender en melding som skal starte en activity, så må jeg være sikker på at noden faktisk har denne og kan starte den.
- Først må vi koble opp en GoogleApiClient
- Deretter bruker vi MessageAPI til å sende en melding. Denne meldingen må sendes asynkront og kan ikke gjøres på UI-tråden. Kan enten hente ut ResultCallback, eller manuelt flytte operasjoen til en bakgrunnstråd og så kalle await() (som blokker)

Message API
- mottaker

- For å lytte på meldinger må mottaker registrere en MessageListener:
  Wearable.MessageApi.addListener(…)

```
@Override
public void onMessageReceived(MessageEvent messageEvent) {
    if (messageEvent.getPath().equals(UNIQUE_PATH)) {
        // Gjør noe, f.eks start en Activity
    }
}
```

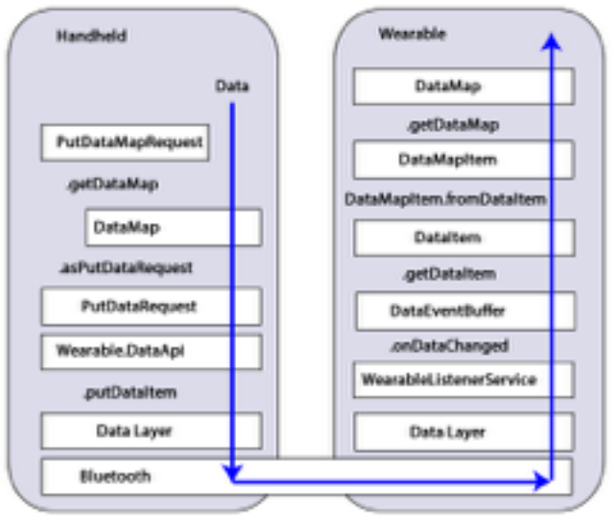
- MessageListener vil motta meldingene. Det er da opp til lytteren å sjekke path for å finne ut hva den skal gjøre med meldingen.
- Mottaker er forøvrig en Activity

DataItem

- Synkronisering av data mellom to enheter
- **DataItem**
  - Dataobjekt bestående av en path og payload
- **DataMap**
  - Wrapper et DataItem-objekt
- **DataMapItem**
  - Wrapper et DataMap-objekt
- **PutDataRequest**
  - Request-objekt brukt for kommunikasjon
- **PutDataMapRequest**
  - Request-objekt som holder på Data Items

- Ser de involverte typene her
- DataMap forenkler serialisering av DataItems, da den tolker DataItems som Bundles

DataItem



Figur lånt fra http://android-wear-docs.readthedocs.org/

- Mer omfattene enn Message API
- Ser at det er brukt en WearableListenerService i figuren, der vi ellers ville hatt en DataListener
  Forskjellen er at ved å bruke en WearableListenerService er man ikke avhengig av at en Activity kjører

DataItem
- avsender

```
private void syncItems(final ArrayList<DataMap> dataMaps, final String path) {
       PutDataMapRequest putDataMapReq = PutDataMapRequest.create(path);
       putDataMapReq.getDataMap().putDataMapArrayList("items", dataMaps);
       PutDataRequest putDataReq = putDataMapReq.asPutDataRequest();
       PendingResult<DataApi.DataItemResult> pendingResult =
               Wearable.DataApi.putDataItem(googleApiClient, putDataReq);
       pendingResult.setResultCallback(new ResultCallback<DataApi.DataItemResult>() {
           @Override
           public void onResult(DataApi.DataItemResult dataItemResult) {
               if (dataItemResult.getStatus().isSuccess()) {
                   Log.i("syncItems", "SyncMessage successfully sent from Handheld activity");
               } else {
                   Log.i("syncImtes", "Error sending syncMessage from Handheld activity");
               }
           }
       });
   }
```

- Som avsender bygger man opp et request som skal sendes. Dersom tilkoblingen blir borte blir dataene cachet til den er oppe igjen.

DataItem
- mottaker

- Mottaker må registrere en DataListener for å lytte til endringer. Dette gjøres ved å kalle: *Wearable.dataApi.addListener(…)*

```
@Override
 public void onDataChanged(DataEventBuffer dataEvents) {
     for (DataEvent dataEvent : dataEvents) {
         if (dataEvent.getType() == DataEvent.TYPE_CHANGED) {
             DataItem item = dataEvent.getDataItem();
             if (item.getUri().getPath().equals("/getItems")) {
                 DataMap dataMap = DataMapItem.fromDataItem(item).getDataMap();
                 ArrayList<DataMap> dataMapItems = dataMap.getDataMapArrayList("items");
                 // Gjør noe med dataMapItems
             }
         }
     }
 }
```

- Når man har registrert en lytter, vil man få syncmeldinger inn i onDataChanged.
  Må her sjekke hva slags event som skjedde og om path stemmer mtp avsender.

Notifications

- Bridged og Contextual
- Flere actions

- Kanskje den mest sentrale komponenten i Android Wear
- Bridged: Pushes automatisk fra telefonen, som når man feks får melding
- Contextual: Situasjonsavhengige notifications som oppstår ved hendelser, feks som nevnt tidligere hvor mye trafikk det er på vei hjem fra jobb

Notifications

```
Intent viewIntent = new Intent(this, MainActivity.class);
// Lag et PendingIntent med Activity som åpnes ved trykk på "open on phone"
PendingIntent viewPendingIntent =
    PendingIntent.getActivity(this, 0, viewIntent, PendingIntent.FLAG_CANCEL_CURRENT);
NotificationCompat.Builder notificationBuilder =
    new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.icon)
        .setLargeIcon(BitmapFactory.decodeResource(
            getResources(), R.drawable.icon))
        .setContentTitle("Tittel")
        .setContentText("Godset fra Drammen")
        .setAutoCancel(true)
        .setContentIntent(viewPendingIntent);
NotificationManagerCompat notificationManager =
    NotificationManagerCompat.from(this);
        notificationManager.notify(NOTIFICATION_ID, notificationBuilder.build());
```

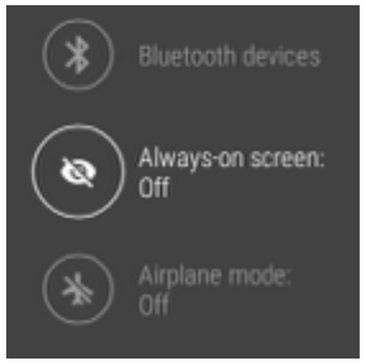
- Lager en PendingIntent som beskriver hva som skal skje når brukeren trykker “åpne på telefon”
  En PendingIntent holder på en komponent (feks Intent) som kan brukes fra en annen app
- Sender så notification på samme måte som man gjør i vanlig mobilapp, ved å bruke NotificationManager.
- Kan spesifisere tittel og innhold

Lister

- WearableListView
  - Viser tre elementer av gangen, med fokus på det i midten
  - *WearableListView.OnCenterProximityListener*
  - Adapter med RecyclerView
  - Klikk ved å implementere *WearableListView.ClickListener*

- For lister bruker vi noe som heter WearableListView, som basically er et ListView som bruker et RecyclerView for caching
  Dette viewet har metoder for å feks se hvilket element som er i “midten”, slik at man kan velge å forstørre dette/endre farge osv.
- Kan lytte på klikk veed å lytte på WearableListView.ClickListener

Lister




```
WearableListView listView = (WearableListView) stub.findViewById(R.id.wearable_list);
listView.setAdapter(new MyListAdapter(items, MainActivity.this));
```

- Her ser vi noen eksempler på hva lister kan brukes til
- Ser under bildene hva som skal til for å opprette et listeobjekt. wearable_list er da deklarert i en eller annen layoutfil
  Deklarer så en adapter

Lister

```java
public class MyListAdapter extends WearableListView.Adapter {
    private final List<Item> items;
    private final Context context;

    public MyListAdapter(List<Item> items, Context context) {
        this.items = items;
        this.context = context;
    }

    @Override
    public WearableListView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(context).inflate(R.layout.todo_item_layout, parent, false);
        return new MyViewHolder(view);
    }

    @Override
    public void onBindViewHolder(WearableListView.ViewHolder holder, int position) {
        MyViewHolder myViewHolder = (MyViewHolder) holder;
        Item item = items.get(position);
        myViewHolder.textView.setText(item.getContent());
    }

    @Override
    public int getItemCount() {
        return items.size();
    }

    public static class MyViewHolder extends WearableListView.ViewHolder {
        public TextView textView;

        public MyViewHolder(View itemView) {
            super(itemView);
            textView = (TextView) itemView.findViewById(R.id.text);
        }
    }
}
```

- En listeadapter bruker man sammen med et WearableListView for å beskrive hvordan dataene skal vises, og bruker en egen XML-fil for å rendre hver rad.

Oppgaver!

http://bekk.github.io/android-wear-kurs/
https://github.com/bekk/android-wear-kurs

- Oppgaver som går det vi har gjennomgått nå
- Eksempelkode ligger på Git