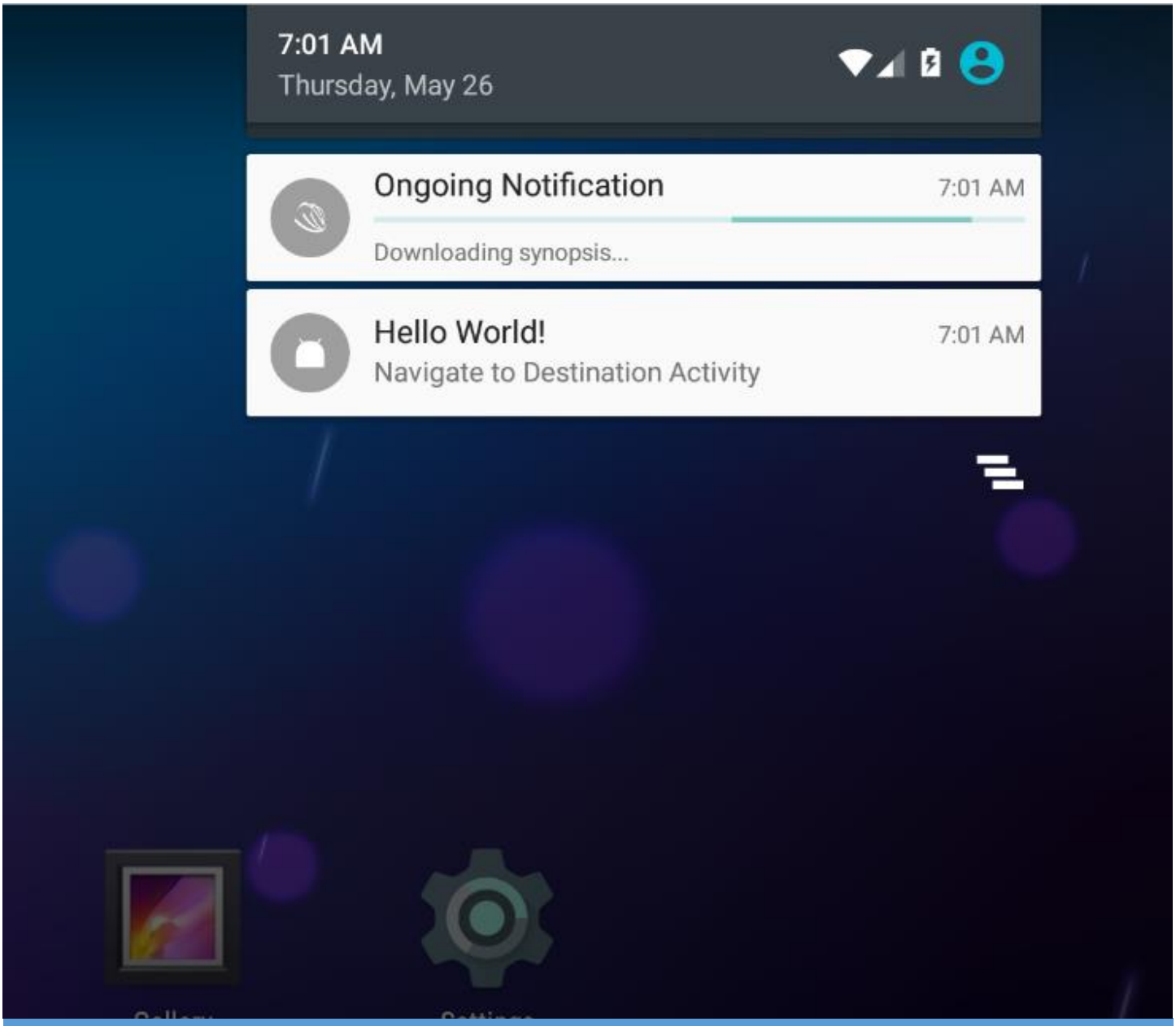


Notifikationer i Android

4. Semester synopsis

23.569 anslag = 9 normalsider

Danny Nielsen 27/5/2016 Erhvervsakademiet Sjælland

Indholdsfortegnelse

Introduktion

Applikationer til smartphones er blevet et kæmpestort marked der hele tiden udvider sig. Hvis man har en original idé til en app, og kan udføre den ordentligt, så kan man leve på fortjenesten. Derfor er det meget vigtigt at holde sig i front på app-markedet. Det gør man blandt andet ved at oprette apps der er informerende når der er brug for det.

Det er her notifikationer kommer ind. Notifikationer er en ekstremt nyttig måde at holde brugeren informeret på. Hvad enten det er ens yndlingskunstner der har udgivet en ny sang, hvornår kagen har fået nok tid i ovnen eller om man har fået en ny SMS eller e-mail. Mulighederne er uendelige.

Jeg vil undersøge notifikationer til Android. Jeg vil undersøge hvordan de virker, hvordan de kan bruges (samt hvornår de burde bruges) og hvilke forskellige slags notifikationer det er muligt at implementere. Det er nemlig vigtigt at tænke over hvordan notifikationerne påvirker brugerne af ens applikation.

Problemformulering

Hvordan implementeres moderne Notifikationer i en Android applikation?

1. Hvorfor er Notifikationer nyttige, og hvad kan de bruges til?
2. Hvilke slags Notifikationer kan implementeres, og hvad er forskellen på disse? (Standard/Ongoing/Insistent/Push)
3. Hvornår burde man anvende Notifikationer i sin applikation? (Og hvornår er de bare overflødige/irriterende?)

Metode

I dette afsnit vil jeg forsøge at redegøre for, hvilke metoder jeg vil tage i brug for at besvare problemformuleringen bedst muligt. Dette kommer til at indebære:

- *Research* - læsning af forskelligt relevant materiale - både i bøger og på internettet.
- *Udførelse* - Udføre, samt eksperimentere med, research-materiale i praksis. Dette vil styrke min indlæringsproces. Jeg vil teste funktioner enkeltvis, og endeligt lave én til flere prototype-applikationer der indeholder relevant funktionalitet til besvarelse af min problemformulering.
- *Synopsisskrivning* - Dokumentering af research- og udførelsesprocesserne. Her vil jeg gå i dybden med hvad jeg har læst, hvad jeg har fundet specielt interessant, samt eksempler på den kode jeg har skrevet og hvilke resultater jeg får af denne.

Planlægning

I dette afsnit vil jeg forsøge at sætte estimater på den tid jeg vil bruge på de førnævnte metoder, som samlet set udgør aktiviteterne:

- "Find the answers"
- "Write the synopsis"

Research bliver den første del af processen. Her vil jeg bruge 1-2 dage på at læse relevante sektioner omkring min problemstilling i bøger og på nettet. Jeg vil i første omgang forsøge at læse materiale der afdækker så meget af min problemformulering som overhovedet muligt.

Udførelsen kommer til at foregå efter *Research*-aktiviteten, hånd i hånd med *Synopsisskrivning*. Disse aktiviteter kommer estimeringsmæssigt til at vare længst tid. Dvs. fra slutningen af *Research* og frem til deadline.
Det kan ikke udelukkes at mere research sideløbende kommer til at være nødvendigt ved enkelte tilfælde. Her er idéen at lade være med at bruge for meget tid på "unødvendig" funktionalitet, men derimod kun researche/implementere funktionalitet der kan hjælpe med at besvare min problemformulering.

Notifikationer – hvorfor er de nyttige?

Notifikationer er efterhånden flittigt brugt på næsten alle platforme – og med god grund. Med alle de enheder vi har i dag, har vi nærmest adgang til en uendelig informationskilde, og vi kan bestemme lige præcis hvad vi har lyst til at få at vide. Er der meget trafik på vej til arbejde i dag? Har dit yndlings fodboldhold scoret et mål? Hvordan bliver vejret de næste par dage? Har du modtaget en besked på de sociale medier? Alt dette og meget, meget mere er det muligt at blive gjort opmærksom på via sin telefon. Hvordan? Primært ved brug af Notifikationer. Med alle de forskellige applikationer der er adgang til, ville det være yderst uproduktivt at være nødsaget til at åbne og tjekke hver og en manuelt efter opdateringer. Her kommer Notifikationer og redder dagen ved altid at sørge for at de vigtigste informationer og opdateringer dukker op når der er brug for dem, også når den pågældende applikation ikke nødvendigvis er åben. Der findes flere forskellige typer Notifikationer, samt flere forskellige måder at anvende dem på[1]. Udover det er der meget specifikke guidelines til hvornår det det er smart at implementere Notifikationer, samt hvornår det ikke er. Dette vender vi tilbage til[2].

[1] Se afsnittene "Min Første Notifikation" & "Ongoing og Insistent Notifikationer"
[2] Se afsnittet "Hvornår skal Notifikationer ikke benyttes? "

Research (interessepunkter)

I denne del af processen læste jeg i tre bøger der indeholdte afsnit omkring notifikationer[3]. Her fik jeg en god del viden omkring notifikationer i Android generelt, samt den generelle syntaks og de vigtigste APIs til oprettelse af notifikationer.

De opdagelser jeg fandt vigtigst er:

Notifikationer er i stand til at[4]:

- Vise et statusbar ikon
- Blinke LED
- Vibrere telefonen
- Afspille notifikationslyde
- Vise ekstra information direkte fra notification tray[5].
- Oprette Intents på notifikationen, direkte fra notification tray.
- Og meget mere…

Overstående er kun en brøkdel af hvad det er muligt at gøre med notifikationer, men det er dog en samling af de mest essentielle og brugte funktioner. Oftest behøver en notifikation ikke at kunne mere.

Ud over dette fik jeg dannet en liste over de vigtigste klasser samt metoder til når der skal implementeres notifikationer:

`NotificationManager`: oprette nye, redigere eksisterende og annullere nuværende notifikationer.

Vigtige metoder indebærer:

```
//Triggers the Notification
notificationManager.notify(0, simpleNotification);

//When the Notification is over
notificationManager.cancel(0);
```

`notify(id, notification)` – Brugt til at trigger/oprette en notifikation.

`cancel(id)` - Brugt til manuelt at afslutte en aktiv notifikation.

[3] Se de første tre bøger i Litteraturlisten.
[4] Meier, R. (2012). Professional Android 4 Application Development. Indianapolis: John Wiley & Sons.
[5] Notifikationsområdet på Android, der kan trækkes ned fra toppen af skærmen.

`Notification`-objektet: objektet der repræsenterer selve notifikationen, og som indeholder den unikke måde notifikationen skal opføre sig på. Alle notifikationens indstillinger er angivet som fields i klassen

Eksempler:

- `notification.sound`
- `notification.vibrate`
- `notification.ledARGB = color.GREEN;`

`Notification.Builder`: en lettere, mere praktisk måde at konfigurere indstillinger/fields til en notifikation blev introduceret i API 11 i form af `Notification.Builder` - i stedet for at angive en masse fields på `notification`-objektet manuelt, kan en Builder gøre arbejdet lettere.[6]

`Notification.Builder` følger samtidig det generelle Builder pattern. Builder pattern er blandt andet blevet opfundet for at gøre objektoprettelser lettere samt mere overskuelige at læse. Hvis en klasse bare har eksempelvis fem forskellige attributter, og der skal være mulighed for at undgå enkelte af dem, så skal der laves en masse forskellige constructors, og objektoprettelse bliver besværligt og ikke særlig pæn kode. Her er Builder pattern en stor hjælp, da det er muligt nemt og enkelt at angive hvilke fields der skal angives på et objekt, og i hvilken rækkefølge.

`PendingIntent`: en fremtidig `Intent` der sender brugeren hen på en Activity hvis notifikationen bliver trykket på.[7]

Men hvorfor ikke bare bruge et "normalt" `Intent`? Forskellen på et `PendingIntent` og et normalt `Intent` er, at hvis et normalt `Intent` navigerer til en fremmed applikation, så udfører den handlinger der bruger den fremmede applikations *Permissions*[8]. Hvis et `PendingIntent` derimod gives til en fremmed applikation, udfører den handlinger der bruger ens egens applikations Permissions. I dette tilfælde er den fremmede applikation `NotificationManager`, som håndterer start, stop og interaktioner med Notifikationer. `NotificationManager` kræver applikationens tilladelser før den kan udføre de handlinger

[6] Se eksempler i afsnittet ”Min Første Notifikation”.
[7] Se mere I afsnittet “Udvidet funktionalitet – Action”.
[8] Tilladelser i Android. Visse handlinger i en applikation kræver brugerens tilladelser, som f.eks. mulighed for internetadgang eller behandling af kamera og systemfiler osv.

som applikationen kræver. Derfor kræver et `Notification`-objekt et `PendingIntent` i stedet for et normalt `Intent`.

Dette var blot et hurtigt overblik over de vigtigste klasser og metoder. Yderligere info som hvordan koden virker, samt hvad der sker hvornår, bliver afdækket ved kodeeksemplerne.

Min Første Notifikation

Så er det blevet tid til det praktiske – at implementere min første notifikation. Det bliver en meget simpel notifikation uden actions. Den skal udelukkende bruges til at afprøve og blive bekendt med Notification-klasserne.

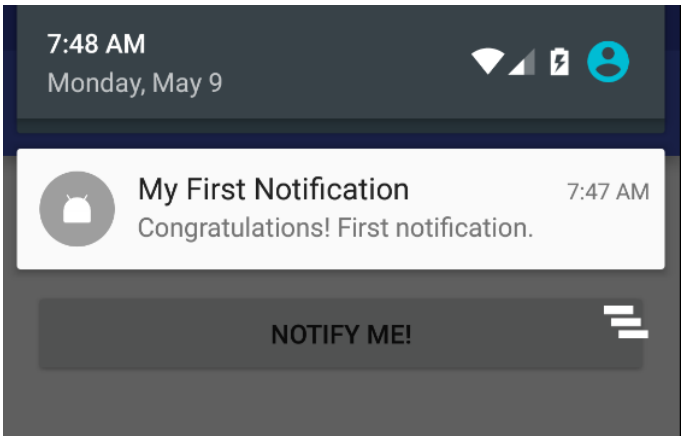
Her er koden:

```
Notification simpleNotification = new Notification.Builder(MainActivity.this)
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentTitle("My First Notification")
        .setContentText("Congratulations! First notification.")
        .setDefaults(Notification.DEFAULT_ALL)
        .setAutoCancel(true)
        .build();

NotificationManager notificationManager = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);

notificationManager.notify(0, simpleNotification);
```

Først bliver et `Notification`-objekt oprettet ved hjælp af `Notification.Builder` klassen.
`Notification.Builder` hjælper med at angive fields hurtigt og nemt på en Notifikation. I dette eksempel er der følgende angivet:

1. `setSmallIcon`: det lille ikon der bliver vist i statusbaren på Android-enheden der angiver en ventende notifikation i Notification Drawer.
2. `setContentTitle`: overskriften på notifikationen.
3. `setContentText`: beskrivelsen på notifikationen.
4. `setDefaults`: her kan man angive standardværdier til enten lyde, vibrationer og LED. I overstående eksempel er alle standardværdier valgt på én gang.
5. `setAutoCancel`: fjerner automatisk notifikationen når brugeren har gjort brug af notifikationens Action. Der er ikke tilføjet nogen Action på Notifikationen endnu, så indtil videre er den property overflødig. Vi vender tilbage til den.
6. `build()`: samler de angivende data fra Builderen sammen, og producerer et `Notification`-objekt med de korrekte indstillinger.

Derefter bliver `NotificationsManager` instantiated, og der er derefter mulighed for at kalde `notify()`–metoden der opretter Notifikationen. Der skal angives et id til den oprettede Notifikation, samt hvilken Notifikation der skal oprettes. Id'et bliver brugt til eventuelle fremtidige opdateringer af samme notifikation. Resultatet ser således ud:



Så simpelt kan det gøres. Det er ikke mange linjer kode der er krævet for at lave en fuldt funktionel Notifikation fra sin app.
Dog er der ikke så meget funktionalitet bag Notifikationen som den er nu. Derfor bliver næste skridt at tilføje en meget nyttig funktion; nemlig en Action.

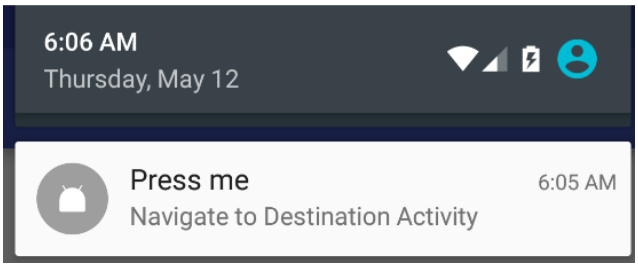
Udvidet funktionalitet – Action

En simpel Notifikation er blevet lavet, men den er ikke så spændende som den er nu. En af de vigtigste funktioner i en Notifikation er evnen til at navigere hen til applikationen som Notifikationen er afsendt fra, samt at blive sendt hen til det rigtige sted i applikationen. Dette gøres ved at tilføje et `PendingIntent`-objekt til Notifikationen.

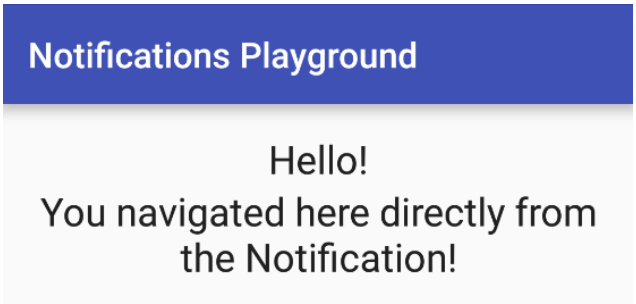
Et `PendingIntent` skal bruge et normalt `Intent` ved oprettelsen. Så her bliver der først lavet et helt normalt `Intent` der sender brugeren hen til en ny Activity. Derefter bliver et `PendingIntent` oprettet, og det bliver gjort via den statiske `getActivity`-metode. Derefter skal der angives en *context*, en *requestcode*, et *intent* og et *flag*. Det angivende flag bruges da vi kun har én `PendingIntent` aktiv ad gangen i denne applikation:

```
//Creating the PendingIntent
Intent intent = new Intent(MainActivity.this, DirectionActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(MainActivity.this, 0,
intent, PendingIntent.FLAG_UPDATE_CURRENT);

Notification simpleNotification = new Notification.Builder(MainActivity.this)
        .setContentIntent(pendingIntent) //The PendingIntent is set here.
        .setContentText("Navigate to Destination Activity")
        //...
```

Resultatet af den nyoprettede Notifikation ser sådan ud:



Og når den trykkes på Notifikationen bliver vi sendt til den angivne Activity:



Det var heller ikke mange linjer kode der var nødvendigt før Notifikationen nu er blevet interaktiv. Dette var en illustration af nogle af de vigtigste byggesten når der skal implementeres Notifikationer i en applikation. I mange tilfælde er denne form for Notifikation nok. Men vi vil også gerne have lov til at kigge lidt på Ongoing og Insistent notifikationer, så forskellen mellem disse er tydeligere og der dermed ikke opstår tvivl om hvilken slags Notifikation der passer bedst til ens app.

Ongoing og Insistent Notifikationer

Den helt standard Notifikation der lige er blevet bygget er blandt de mest brugte. Her er formålet for det meste at underrette brugeren om en ny hændelse, brugeren trykker på Notifikationen og bliver sendt til den nye hændelse i applikationen hvorefter notifikationen forsvinder. Men der findes to andre typer Notifikationer – Ongoing og Insistent.

Ongoing

Ongoing notifikationer bruges i de fleste tilfælde til at informere en bruger om en bestemt igangværende proces som f-eks. Media playback og/eller download. Notifikationer der er Ongoing kan derfor i de fleste tilfælde ikke fjernes af brugeren, men forbliver i Notification drawer til processen er fuldført.

Lad os prøve at oprette en Ongoing Notifikation der simulerer en igangværende proces:

```
Notification ongoingNotification = new Notification.Builder(MainActiv-
ity.this)
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentTitle("Ongoing Notification")
        .setContentText("Downloading...")
        .setProgress(100, 50, true) //Simulates active download.
        .setOngoing(true) //Makes the Notification ongoing.
        .build();

notificationManager.notify(1, ongoingNotification);
```

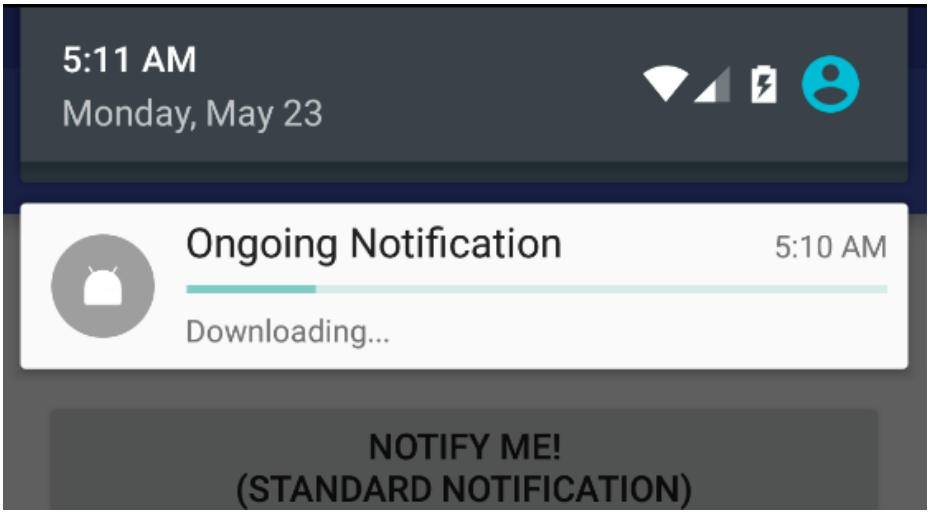
Koden ser meget ens ud i forhold til standard Notifikationen, men med to meget vigtige forskelle:

`setOngoing` og `setProgress`. Her er den vigtigste `setOngoing`, da det er metoden der sørger for at Notifikationen ikke kan fjernes af brugeren, men at den derimod skal afsluttes manuelt ved at bruge:

```
notificationManager.cancel(1);
```

`setProgress`-metoden sørger for at simulere en aktiv download. Det første parameter bestemmer hvornår download er færdig, det andet parameter bestemmer hvor langt download er nået. Disse to er i dette tilfælde ligegyldige, da det tredje parameter bestemmer at der bliver simuleret download indtil notifikationen bliver fjernet manuelt.

Resultatet ser således ud:



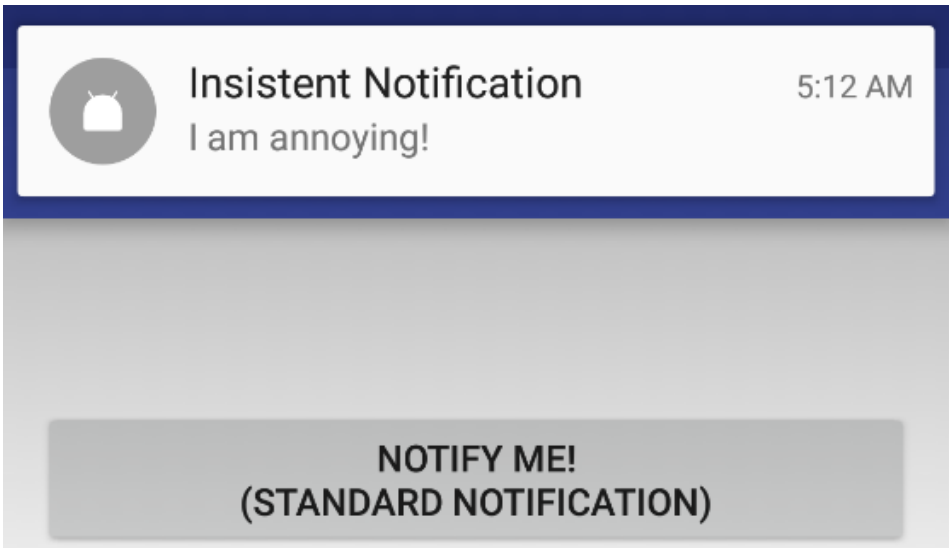
Insistent

Insistent notifikationer er notifikationer der bliver ved med at gentage lyd, lys og vibration indtil de bliver afsluttet af brugeren. Gode eksempler på dette kan være telefonopkald eller en alarm. I normale tilfælde ville man ikke bruge Insistent Notifikationer i tredjepartsapplikationer, da denne type Notifikationer kan være meget forstyrrende for brugeren.

Dette er også afspejlet af manglen på en metode i Notification Builderen der kan sætte dette flag. Det skal sættes manuelt på `Notification`-objektet:

```
Notification insistentNotification = new Notification.Builder(MainActiv-
ity.this)
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentTitle("Insistent Notification")
        .setContentText("I am annoying!")
        .setPriority(Notification.PRIORITY_HIGH)
        .setSound(RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFI-
CATION))
        .setDefaults(Notification.DEFAULT_LIGHTS | Notification.DEFAULT_VI-
BRATE)
        .build();

insistentNotification.flags = insistentNotification.flags | Notifica-
tion.FLAG_INSISTENT;

notificationManager.notify(2, insistentNotification);
```

Her bliver der tilføjet lyd, lys og vibration til Notifikationen ved hjælp af `setSound` og `setDefaults`-metoderne. Notifikationen angives som Insistent til sidst i `flags`-attributten.



Ydermere dukker notifikationen op fra toppen og ”forstyrrer” den nuværende aktivitet.

Push Notifikationer

En sidste type notifikation der bestemt er værd at nævne, er de såkaldte Push notifikationer. Disse Notifikationer er specielle på den måde, at deres data bliver sendt fra en server og ned til de individuelle applikationer der derefter opretter en notifikation ud af denne. Dette kræver at applikationen benytter en bagvedliggende tredjepartsservice der udsender data til alle brugerne af denne app. Derfor er det nødvendigt at registrere de individuelle enheder der bruger appen til Push i webservicen.

Den foretrukne service til brug af Push-beskeder i Android er Google Cloud Messaging (GCM). En tredjepartsservice bliver oprettet som kommunikerer med GCM. Udover det, kræves det at den oprettede webservice håndterer logikken bag beskederne der skal sendes. GCM står alene for at videresende beskeder fra webservicen til den/de Android enheder det vedrører.

Dette kræver også arbejde fra Android-siden. Android applikationen står nemlig for at formidle den tilsendte data og fremvise denne som Notifikationer til brugeren. Men hvad hvis telefonen er på standby, slukket eller hvis internetadgang er slået fra? Hvordan sørger man for at Notifikationen når frem alligevel? Her kommer Googles GCM API i spil. Med denne er der mulighed for at vække telefonen fra standby i få sekunder og tjekke om der er nye beskeder ventende fra serveren. APIen benytter enhedens *Wake Lock*[9] til dette. Hvis GCM ikke kan komme i kontakt med telefonen, lagrer den de ventende beskeder til telefonen igen er tilgængelig.

[9] En batterieffektiv service i Android systemet der kan vække enheden fra Standby i intervaller. Dette gør det muligt at tjekke efter opdateringer, beskeder mm. selvom telefonen ikke er i brug.

Den måde det foregår på, er at lave en klasse der udvider `GcmListenerService`-klassen. Efterfølgende handler det bare om at override `onMessageRecieved()`–metoden:

```
public class GcmIntentService extends GcmListenerService{

    @Override
    public void onMessageReceived(String from, Bundle data) {
        String message = data.getString("message");
        NotificationManager notificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
        Notification pushNotification = new Notification.Builder(this)
                .setSmallIcon(R.mipmap.ic_launcher)
                .setContentTitle("Test")
                .setContentText(message)
                .build();

        notificationManager.notify(5, pushNotification);
    }
}
```

Så handler det bare om at læse den data som GCM har fået fra tredjepartswebservicen der ligger i `Bundle`-variabelen i parameteret.

Før beskeden kan nå til de korrekte enheder, er det vigtigt at registrere hver enhed hos GCM, så GCM ved hvilke specifikke Android-enheder dataene skal sendes til. Det foregår således:

```
InstanceID instanceID = InstanceID.getInstance(getBaseContext());
String token = instanceID.getToken("258408041486", GoogleCloudMessaging.IN-
STANCE_ID_SCOPE, null);
```

Det dannede token er unikt for hver enhed, og kan dermed registreres hos GCM. Når tredjepartsservicen får oplyst det/de unikke tokens per enhed, så kan der blive sendt beskeder til disse fra webservicen.

En sidste ting der er værd at nævne før Push er muligt, er Permissions. Permissions er som førnævnt rettigheder i Android applikationer. Der skal angives i applikationens *Manifest*-fil at applikationen har adgang til at benytte internet, Wake Lock samt at kunne modtage og behandle beskeder fra GCM:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
<permission
android:name="com.dda.notificationplayground.permission.C2D_MESSAGE"
android:protectionLevel="signature" />
<uses-permission
android:name="com.dda.notificationplayground.permission.C2D_MESSAGE" />
```

Google har gjort det forholdsvist let at implementere Push-meddelelser i sin Android notifikation ved hjælp af GCM. Den største udfordring er, at skabe en webservice der kan

håndtere beskeder fra telefonen og sende dem videre til GCM. Dog findes der rigtig mange trejdepartsservices det er muligt at benytte sig af, så det ikke er krævet at bygge sin helt egen.

Hvornår skal Notifikationer ikke benyttes?

Indtil videre har vi kigget både på hvorfor Notifikationer i Android er nyttige, hvilke slags Notifikationer det er muligt at implementere, samt hvordan disse implementeres i praksis. Jeg nævnte kort tidligere[10] at der er meget specifikke guidelines til hvornår der burde bruges Notifikationer, samt hvornår der *ikke* burde. Google har et bestemt pattern til implementering af Notifikationer i Android. Dette vil jeg tale om til eksamen.

Konklusion

Min problemformulering indeholder ét overspørgsmål og tre underspørgsmål.
Hovedspørgsmålet lyder således:

"Hvordan implementeres moderne Notifikationer i en Android applikation?"

Og dette spørgsmål har jeg forsøgt at besvare ved at besvare de tre underspørgsmål:

1. Hvorfor er Notifikationer nyttige, og hvad kan de bruges til?
2. Hvilke slags Notifikationer kan implementeres, og hvad er forskellen på disse? (Standard/Ongoing/Insistent/Push)
3. Hvornår burde man anvende Notifikationer i sin applikation? (Og hvornår er de bare overflødige/irriterende?)

Svarene til disse spørgsmål har jeg fundet frem til dels ved hjælp af research, og dels ved udførelse af materiale i praksis. Uddybende svar er blevet angivet igennem denne synopsis. For at opsummere:

1) Notifikationer er en aldeles praktisk måde at komme i kontakt med brugerne af ens applikation. Det kan ikke forlanges at brugeren konstant sidder klar med sin enhed for at tjekke om ny data er tilgængelig. Det ville være særdeles upraktisk. I stedet er der mulighed for at underrette applikationens brugere mens enheden ligger i lommen, eller er slukket, ved brug af Notifikationer. Det er efterhånden blevet svært at finde applikationer der ikke benytter sig at Notifikationer, og med god grund. Notifikationer bliver nemlig brugt til nærmest alt: sportsopdateringer, vejropdateringer, trafikopdateringer, påmindelser, underretninger om

[10] Se afsnittet ” Notifikationer – hvorfor er de nyttige? ”

SMS/e-mail/opkald, opdateringer fra de sociale medier, navigationsruter, musikhåndtering, enhedshåndtering og meget, meget mere.

2) Det er blevet fastlagt at der er mange forskellige måder at bruge Notifikationer på. Derfor er det også muligt at oprette forskellige slags Notifikationer til forskellige formål. Det er for eksempel ikke altid nok at afsende en "standard" notifikation der bare viser et ikon, en overskrift og en tekst. Nogle gange vil man gerne vise at en bestemt proces er i gang, eller have mulighed for aktivt at interagere med Notifikationen.

Der findes tre forskellige Notifikationer der kan implementeres:

- **Standard** – En helt normal Notifikation der som minimum skal bestå af et ikon, en overskrift og en tekst. Denne form for Notifikation har oftest til formål at underrette brugeren, hvorefter brugeren kan trykke på Notifikationen og blive sendt ind i applikationen der har sendt denne.
- **Ongoing** - En Notifikation der ikke kan fjernes manuelt af brugeren. Siden det ikke er muligt at fjerne Notifikationen, har denne form for Notifikation oftest til formål at informere om en nuværende status på enheden, eller eksempelvis en igangværende download. Når status på enheden ændrer sig, eller download er færdig, forsvinder Notifikationen automatisk. Et eksempel kan være når en Android-enhed tilsluttes en computer. Her vil en aktiv Ongoing Notifikation informere om hvilken tilslutningstype der er tale om, indtil Android enheden igen fjernes fra computeren.
- **Insistent** - En Notifikation der primært har til formål at underrette brugeren om noget der kræver opmærksomhed med det samme. Derfor gentager denne type Notifikation dens vibrations, lys og lydmønster lige indtil brugeren manuelt ser/interagerer med den. Denne type Notifikation bruges oftest kun til vigtige begivenheder som telefonopkald eller alarmer.

Det er i øvrigt mange forskellige muligheder for at brugerdefinere alle typer Notifikationer. Dermed kan en Notifikation skræddersys lige efter behov. Google har gjort livet lettere for udviklere, og lavet en `NotificationBuilder`-klasse som gør håndteringen af Notifikations-funktioner meget enkel og hurtig[11].

Jeg fik implementeret hver af disse tre typer Notifikationer under udførelsesfasen. Dog er det er svært at fremvise eksempler på de sidste to typer Notifikationer uden en fysisk telefon at se det på. Dette har jeg mulighed for at gøre til eksamen.

[11] Se eksempler i afsnittene ”Min Første Notifikation” & "Ongoing og Insistent Notifikationer"

3) Svaret til dette spørgsmål vil jeg gå i dybden med, og besvare til eksamen.

En anden vigtig ting jeg kan tage med mig efter denne opgave, er den viden jeg har fået omkring Push-meddelelser. Det smarte ved Push meddelelser er, at det ikke nødvendigvis er begrænset til brug i Android applikationer. Push meddelelser er også brugt på eksempelvis Apple og Windows enheder. Implementeringsprocessen er lidt forskellig, men princippet omkring server-til-enhedskommunikation er det samme. Det er et meget vigtigt værktøj til at bygge bedre applikationer.

Refleksion

I dette afsnit vil jeg reflektere lidt over mit arbejde. Efter at have valgt et relevant emne, fik jeg dannet en problemformulering til opgaven. Jeg synes at de opstillede spørgsmål har været gode at følge, og har dannet grundlag for besvarelse både i teori og i praksis.

Min metode fik jeg fulgt som forventet, og jeg føler at dette har været en god måde at gøre det på. Det har sørget for at jeg kunne opdele processen på en god måde ved at blande læsning, programmering og synopsisskrivning.

Planlægningsmæssigt gik det også godt, og jeg fik holdt mig til de estimater jeg havde lavet. Jeg nævnte i planlægningsafsnittet at jeg ville forsøge at holde mig fra ”unødvendige” funktioner så meget som muligt, så jeg ikke ville spilde min tid på irrelevante ting. Dette gik også for det meste godt, men set i bakspejlet kunne jeg godt have nøjedes med at bruge lidt mindre tid på research/udførelse af Push Notifikationer.

Litteraturliste

Meier, R. (2012). *Professional Android 4 Application Development*. Indianapolis: John Wiley & Sons.

Delessio, C. (2015). *Android application development in 24 hours*. INpolis, IN: Sams.

Phillips, B., & Hardy, B. (2015). *Android Programming: The Big Nerd Ranch Guide* (2nd ed.).

Pubnub Developers. *Sending and Receiving Android Push Notifications w/ GCM,* https://www.pubnub.com/blog/2015-06-24-sending-receiving-android-push-notifications-with-gcm-google-cloud-messaging/

Android Developers. *API Guides, Notifications*, http://developer.android.com/guide/topics/ui/notifiers/notifications.html

Android Developers. *Patterns, Notifications,* https://www.google.com/design/spec/patterns/notifications.html#notifications-usage

Android Developers. *Google Cloud Messaging*, https://developers.google.com/cloud-messaging/

Mixpanel Developers, *Sending Android Push notifications from Mixpanel*, https://mixpanel.com/help/reference/android-push-notifications