NEW MEDIA &
COMMUNICATION
TECHNOLOGY

Google Cloud Messaging

MAD - DES-Project

Nicolas Delanghe

2NMCT1

Inhoud

Inleiding

Voor mijn DES-Project heb ik gekozen om een applicatie te bouwen op basis van Google Cloud Messaging (GCM). Dit is een online service die app ontwikkelaars toelaat berichten te versturen naar app-instanties die geïnstalleerd zijn bij hun gebruikers en andersom. De toepassingen hiervoor zijn eindeloos. Het meest voor de hand liggende gebruik hiervoor is voor chatapplicaties. Gebruikers sturen met hun app een bericht naar de server en deze stuurt het door naar de juiste ontvanger of ontvangers.

Dit document bevat een korte introductie tot GCM alsook de nodige uitleg bij mijn bijhorende applicatie: Howest Messenger.

GCM Basis

Wat is GCM?

Google Cloud Messaging is een gratis Cloud service die berichten kan sturen naar client-applicaties. Dit kunnen Android of iOS apps zijn, maar sinds kort is er ook integratie met Chrome. Het is mogelijk om een bericht te sturen van de server naar clients, maar clients kunnen ook zelf communiceren met de server via upstream messages. Belangrijk om op te merken is dat het om kleine berichten gaat van maximum 4kb. Het is dus niet de bedoeling dat gegevensets doorgegevens worden op deze manier, maar eerder dat de client verwittigd wordt dat er gegevens beschikbaar zijn om af te halen. Om GCM te implementeren in eigen applicaties moeten hiervoor de nodige achtergrond services geconfigureerd zijn bij de client en moet er een server zijn die achter de schermen alle trafiek regelt.

Server Side

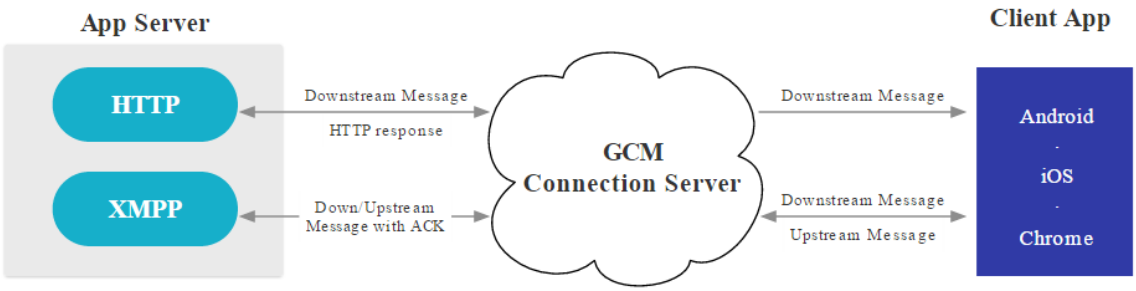
De servers aan de kant van de ontwikkelaar communiceren niet rechtstreeks met de clients. Op die manier moeten zij ook niet over alle gegevens beschikken. Aan de serverkant maakt men het onderscheid tussen de applicatieservers van de ontwikkelaar en de GCM-connectieservers. De applicatieserver heeft een server API-key nodig om zijn calls te verantwoorden. Deze API-key is niet voor gebruik bij de clients en mag daar ook, uit veiligheidsoverwegingen, niet gekend zijn. Clients moeten zich verantwoorden met een ander soort token. De communicatie tussen de servers kan verlopen volgens 2 verschillende protocollen: HTTP (Hypertext Transfer Protocol) of XMPP (Extensible Messaging and Presence Protocol). Elke protocol heeft zijn voor- en nadelen. Er kan natuurlijk ook met beide protocollen gewerkt worden. Als je de nadelen van de protocollen naast elkaar zet, kan je afleiden dat dit meestal ook het geval zal zijn.

HTTP:

- Enkel downstream berichten
- Synchroon (wachten op antwoord)
- Multicast mogelijk
- JSON en Plain Text formaat

XMPP:

- Upstream en downstream berichten
- Asynchroon (sturen zoveel en wanneer je wil)
- Multicast niet mogelijk
- Enkel JSON



Indien een client een bericht wil sturen naar de applicatieserver, stuurt hij een XMPP-bericht naar de GCM-connectieserver. Deze kijkt of de applicatieserver online is. Als dit niet het geval is wordt het bericht bijgehouden tot de applicatieserver weer beschikbaar is.

Client Side

Voordat een GCM-client berichten vanaf de server kan ontvangen, moet hij zich registreren. Om te registreren wordt de Instance ID API aangesproken met als parameter de Sender ID (Project number binnen de Google Developer Console). Het resultaat hiervan is een Instance ID token, dit is een uniek ID dat anders is voor elke instantie van de app. De client stuurt dan zijn net verkregen token naar de applicatieserver waar dit bijgehouden wordt.

Nu kunnen clients aangesproken met broadcast-berichten naar alle clients of via een unicast-bericht die net die ene client aanspreekt op basis van zijn token. Moest het token onderschept worden, wordt deze gewoon gedropt en begint de procedure opnieuw. Indien een server iets wil sturen naar een client die niet beschikbaar is, wordt het bericht bijgehouden op de GCM-connectieservers totdat de client terug online is.

Device Groups en Topics

Clients sturen berichten naar de server en de server stuurt berichten naar de clients. Het is natuurlijk niet interessant dat alle berichten door iedereen ontvangen en gelezen kunnen worden in de context van bijvoorbeeld chatberichten. Om dit op te lossen kunnen client tokens gegroepeerd worden in 2 soorten groepen: Device Groups en Topics. Op deze manier kan de server zeer specifiek bepalen naar welke devices hij berichten stuurt.

Device Groups zijn groepen devices die nauw met elkaar samenwerken, zoals een gsm en tablet van dezelfde gebruiker. Alle devices in een groep hebben dezelfde notification key. Dit is een extra key die gebruikt wordt om de registration tokens van alle devices in groep gezamenlijk aan te spreken. Devices binnen een Device Group kunnen ook sturen naar andere toestellen binnen dezelfde groep. De maximum grootte van een Device Group is 20 toestellen.

Topics zijn onderwerpen waarop een client kan inspelen op basis van het publish/subscribe-model. Een bericht naar een topic heeft als bestemming */topics/TOPICNAAM*. Voor een navigatieapp kan een topic bijvoorbeeld een verkeersknooppunt zijn. Een client abonneert zich dan op deze topic rond de periode van de verplaatsing van de gebruiker. De applicatieserver stuurt berichten naar deze topics en alle clients die binnenkort bij de plaats van de topic moeten zijn worden verwittigd van eventuele problemen. De server gebruikt de Instance ID API's om informatie te weten te komen over clients op basis van hun registration token. Je kan appinstanties laten abonneren op bepaalde topics en opvragen op welke topics een appinstantie geabonneerd is.

Mijn Project: Howest Messenger

Idee

Niet iedereen controleert dagelijks zijn email. In vergelijking met wat er nu allemaal beschikbaar is, is email een lomp, omslachtig systeem dat, zeker bij studenten, uitsluitend gebruikt wordt voor formele communicatie. Al zeker niet iedereen bekijkt dagelijks zijn Leho-notificaties. De site is traag en de kans dat er juist vandaag belangrijke informatie op staat is klein. Er worden voor belangrijke zaken soms e-mails gestuurd, maar als het last minute informatie is, kan deze weleens te laat bekeken worden. Je kan natuurlijk een e-mailapplicatie op je smartphone gebruiken die notificaties geeft, maar niet iedereen kiest hiervoor omdat tegenwoordig het informatie/reclameratio niet echt gunstig is.

De oplossing hiervoor is een applicatie die informatie met een pushmelding tot bij de student brengt. Niet via een e-mail, maar via een ander platform dat uitsluitend bedoeld is voor belangrijke informatie (Lees: de les is verzet). Zo kan de student zonder eigen initiatief of onnodige spam op de hoogte blijven van wat voor hij of zij belangrijk is. Om de scope wat in te perken heb ik me beperkt tot informatie in verband met de NMCT-vakken uit semester 4.

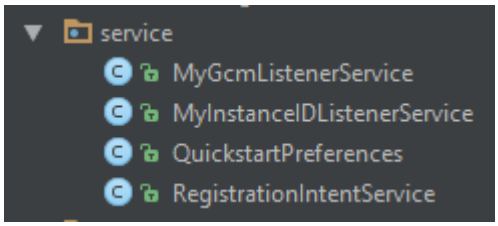
Projectbasis

Voor mijn app ben ik vertrokken van een sample van Google die toelaat downstream berichten te sturen. Ik heb deze code eerst onderzocht, en heb dan een nieuw project aangemaakt met dezelfde package name.

Het project bestaat uit 2 delen: een Android app en een extra project dat een command toevoegt aan de Gradle Terminal. Aan de hand van deze command kan een server geëmuleerd worden. Op deze manier moest ik geen tijd spenderen aan een eigen Ubuntu server, wat mijn originele plan was. De command werkt als volgt:

```
.\gradlew.bat run -Pmsg="BERICHT PAYLOAD"
```

Na onderzoek en veel trial-and-error ben ik uitgekomen bij volgend service-package voor mijn project:



MyGcmListenerService bevat het aanknopingspunt door de rest van mijn project. Deze klasse bevat een methode die opgeroepen wordt als we een bericht ontvangen. Belangrijk om op te merken is dat we geen notificatie krijgen hiervan, tenzij we dit zelf programmeren. Het is dus mogelijk berichten te ontvangen zonder de gebruiker op de hoogte te brengen. In de variabele “message” zit de payload van het eerder verstuurde command. Deze payload zet ik om naar een object en op basis van dit object wordt er een pushmelding gemaakt. Omdat JSON vervelend is om te typen, heb ik een eigen formaat voorzien om berichten te versturen met de terminal. In de praktijk worden berichten ook niet getypt, maar zijn dit JSON-serialisaties van objecten die op een gebruiksvriendelijke manier zijn aangemaakt. Indien het bericht niet voldoet aan het formaat, wordt er niet op gereageerd.

```
.\gradlew.bat run -Pmsg="titel/berichtinhoud/auteur/vaknr"
```

```
@Override
public void onMessageReceived(String from, Bundle data) {
    String message = data.getString("message");
    Log.d(TAG, "From: " + from);
    Log.d(TAG, "Message: " + message);
    ...
}
```

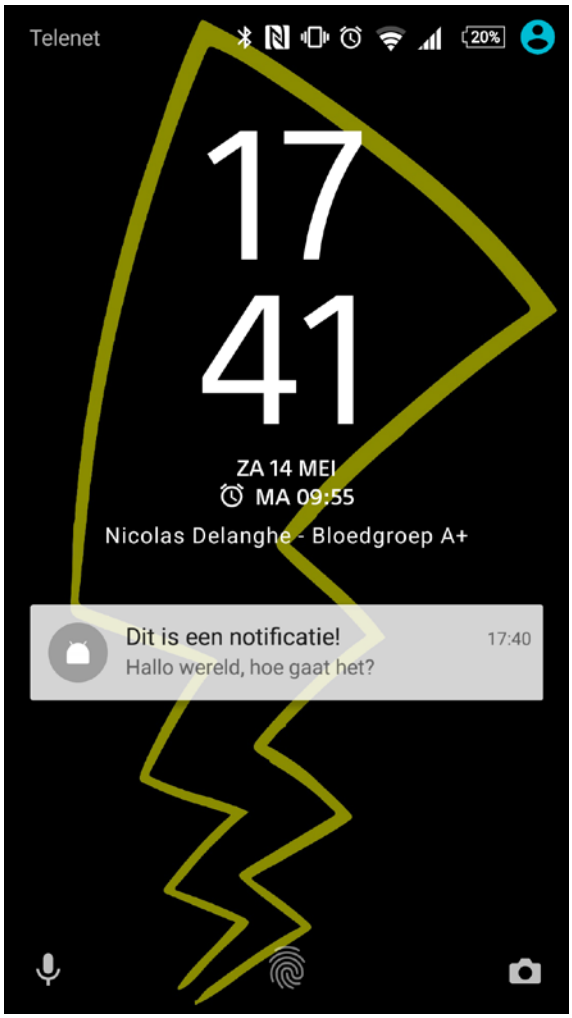
MyInstanceIDListenerService is een service die wacht tot dat het token gerefresht wordt. Deze zal dan een nieuwe service maken via een Intent en deze starten.

QuickstartPreferences bevat enkele String-constanten voor gebruik doorheen de services en de rest van het project.

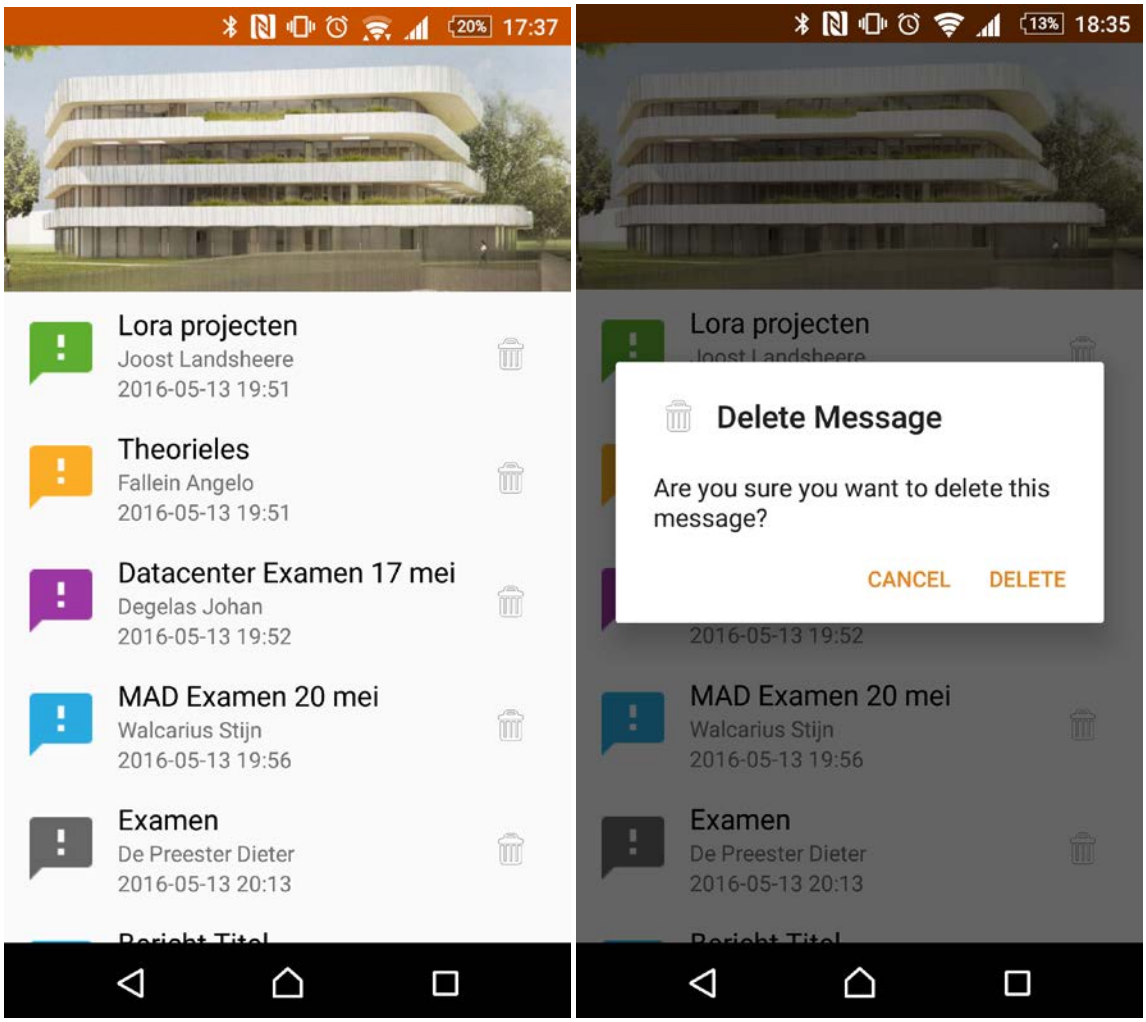
RegistrationInstentService is de service die aangeroepen wordt door MyInstanceIDListenerService. Deze zal een Instance ID token genereren en sturen naar de server.

Verdere uitwerking

Als de gebruiker de applicatie voor de eerste keer start, wordt er met behulp van bovenstaande klasses een token verstuurd naar de server. Indien de app een bericht ontvangt, wordt er een notificatie gemaakt op basis van het toegestuurde bericht. Deze wordt getoond aan de gebruiker:



Aan deze notificatie is een Intent gekoppeld die de applicatie opent en de lijst van berichten toont aan de gebruiker. De lijst is gemaakt in een RecyclerView met bijhorende rowlayout en adapterklasse. Aangezien de berichten niet zo groot zijn, is SharedPreferences een ideale manier om alle berichten op te slaan. Het nadeel is dat de berichten weg zijn als de app verwijderd wordt omdat de databron het GCM-bericht is en niet een server waar de berichten van kunnen opgehaald worden. De lijst bevat een kleine hoeveelheid informatie over de berichten: de berichttitel, de auteur, het vak en het tijdsstip van ontvangst. Berichten die niet meer nodig zijn kunnen verwijderd worden. Ter confirmatie is er een bevestigingsscherm.



Als er op een lijst item getikt wordt, gaat men naar een detailscherm die gemaakt werd door het parcelable object mee te geven. Hier kan de gebruiker de effectieve inhoud van het bericht bekijken en achteraf terugkeren naar de lijst met de back-toets.

Problemen

De grootste beperking van mijn applicatie is dat het bericht verloren gaat als men de notificatie wegswipet. De reden hiervoor is dat het bericht pas opgeslagen wordt als je de app opent via de Intent in de notificatie. Dit kan niet eerder gebeuren omdat er in de achtergrondservice geen context aanwezig is. Deze context is nodig om te kunnen schrijven naar de SharedPreferences. Andere applicaties lossen dit makkelijk op door berichten op te halen van een server bij het openen van de applicatie. Aangezien deze app als enige informatiebron de notificaties heeft, kan ik deze oplossing niet zelf toepassen.

Een minder groot probleem was dat het start project nog van voor het Material Design tijdperk was. De minimum API voor GCM is namelijk 9. Het lukte mij niet om in dit project een layout te verkrijgen die zich gedroeg zoals modernere apps. Zelfs het aanpassen van de kleuren was een ramp aangezien de enige referentie naar de kleuren ogenschijnlijk in een gegenereerde file zaten die ik niet kon aanpassen. Ik kon niet zomaar nieuw project aanmaken aangezien de server dan niet meer zou werken. Dit heb ik opgelost door een project aan te maken met dezelfde packagenaam zodat ik de instellingen van de server zonder veel probleem kon overnemen.

Een laatste probleem was de server zelf. Google biedt enkel de structuur van de request aan die nodig is om een bericht te versturen. Verder is er niets om op verder te bouwen. Mijn enige leidraad was een eenvoudig stappenplan die uitlegde hoe ik een Ubuntu server kon maken die met Python makkelijk berichten in het juiste formaat tot bij de GCM-servers kon krijgen. Ik heb pas later de terminaluitbreiding gevonden die ik uiteindelijk verder heb gebruikt in mijn project.

Bronnen

Cloud Messaging. (z.d.). Geraadpleegd 13 mei 2016, van https://developers.google.com/cloud-messaging/

About GCM Connection Server. (z.d.). Geraadpleegd 13 mei 2016, van https://developers.google.com/cloud-messaging/server

Create Push Notification Server for Android with GCM using Python. (z.d.). Geraadpleegd 14 mei 2016, van https://www.digitalocean.com/community/tutorials/how-to-create-a-server-to-send-push-notifications-with-gcm-to-android-devices-using-python

Device Group Messaging. (z.d.). Geraadpleegd 14 mei 2016, van https://developers.google.com/cloud-messaging/notifications

Google Cloud Messaging: Overview. (z.d.). Geraadpleegd 13 mei 2016, van https://developers.google.com/cloud-messaging/gcm

Registering Client Apps. (z.d.). Geraadpleegd 13 mei 2016, van https://developers.google.com/cloud-messaging/registration

Send Messages to Topics. (z.d.). Geraadpleegd 14 mei 2016, van https://developers.google.com/cloud-messaging/topic-messaging

Set up a GCM Client App on Android. (z.d.). Geraadpleegd 14 mei 2016, van https://developers.google.com/cloud-messaging/android/client

Simple Downstream Messaging. (z.d.). Geraadpleegd 14 mei 2016, van https://developers.google.com/cloud-messaging/downstream

Try Cloud Messaging for Android. (z.d.). Geraadpleegd 14 mei 2016, van https://developers.google.com/cloud-messaging/android/start

What is Instance ID? (z.d.). Geraadpleegd 13 mei 2016, van https://developers.google.com/instance-id/