Praktikum Mobile und Verteilte Systeme

Mobile Push Architectures

Prof. Dr. Claudia Linnhoff-Popien
Michael Beck, André Ebert
http://www.mobile.ifi.lmu.de

Wintersemester 16/17

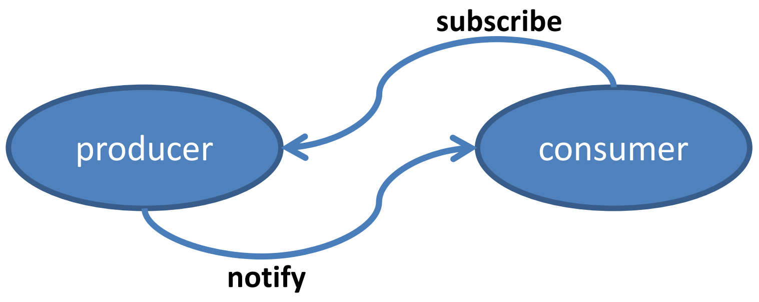
Asynchronous communications

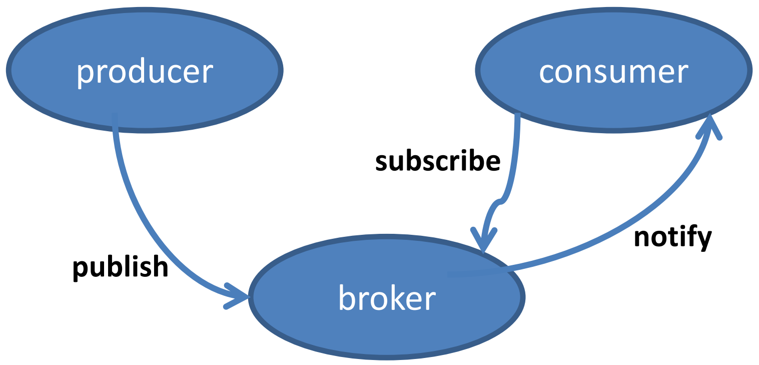
How to notify clients about changed resources or updates?

More general: How to **handle server-side events asynchronously**?

- **polling** is ineffective (e.g., continuously requesting a web service)

- SOAP offers **WS-Notification**
  - either peer-to-peer or brokered





- **Comet programming**: strategies for realizing push-like communication in pull-based environments (using HTTP)

Comet programming

- A web application model using persistent HTTP requests to push data to a browser
- Term coined by software engineer Alex Russell in a blog post in 2006
- First implementations date back to 2000
  - Pushlets, Lightstreamer, KnowNow
- In 2006, some widely known applications adapted these techniques
  - web-based chat application for AOL, Yahoo, Microsoft chat (Meebo)
  - Google: integration of a **web-based chat** in GMail
  - Comet-based, real-time collaborative document editing (JotSpot)

- Comet is an umbrella term, encompassing multiple techniques
  - relying on features included by default in browsers (e.g., JavaScript)
  - also known as **Ajax Push**, **Reverse Ajax**, Two-way-web, HTTP Streaming

Comet implementations

- **Streaming-based** implementations
  - Hidden iframe
    - uses chunked transfer encoding (no content-length) containing JavaScript tags
    - working in every common browser
  - XMLHttpRequest
    - server sends “multipart HTTP response” with each part invoking `onreadystatechange` callback
    - only working with few browsers

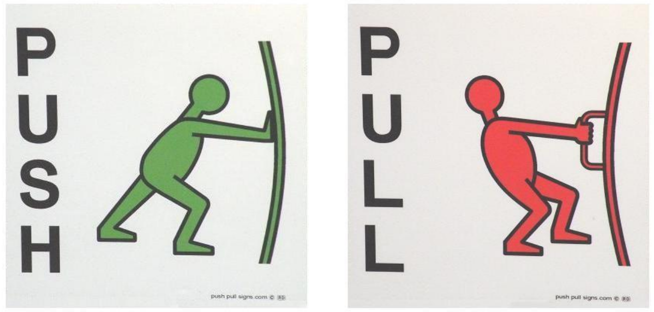
- **Long-polling** based implementations
  - XMLHttpRequest long polling
    - works like the standard use of XHR
    - an asynchronous request is sent to the server, response only after an update
    - after processing the response (or after a timeout), a new request will be sent
  - Script tag long polling
    - dynamically create script elements as `<src="cometserver/…js">`
    - payload contains new JavaScript events
    - cross-browser and cross-domain functionality

Mobile push architectures

- **Push notifications…**
  - are messages pushed to a central location and delivered to mobile devices
  - are comparable to the publish/subscribe pattern
  - often contain other technologies such as alerts, tiles, or raw data
  - offer an alternative to constantly polling data from servers

- These "central locations" are nowadays provided by Google, Apple, Microsoft, Blackberry, …

- Goal: **Push, don't pull**
  - only fetch data when useful



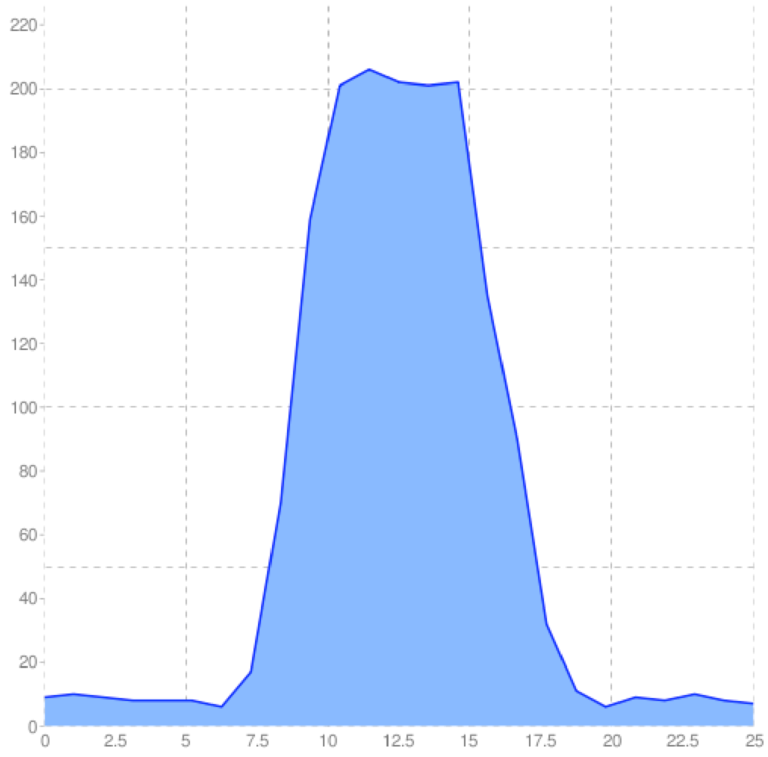
Advantages of push notifications (1)

Battery Life

- Baseline: 5-8 mA
- Network: 180-200 mA
- Radio stays on for few seconds
- 0.50 mAh for a short poll
  - 5m frequency: ~144 mAh / day
  - 15m frequency: ~48 mAh / day

- Push notification services are running in the background

- Pushing data is hence **more effective** than polling, if #updates < #polls



Source: Android development team at Google

Advantages of push notifications (2)

- **Message delivery and „time of flight“**
  - to save on battery, polls are usually spaced **15+ minutes apart**
  - updated data might hence also be **15+ minutes late**!
  - when using push notifications, message delivery can usually be expected to be a matter of seconds (<5s)
  - push notifications can also be sent to a currently offline device

- However, generally there is **no guarantee for delivery**
  - one might **exceed quotas**
  - some notification servers only allow a single message to be in **queue** at a time
  - …

Google C2DM

- The **Cloud to Device Messaging framework** allowed third-party servers to send lightweight messages to corresponding Android apps
- Designed for notifying apps about new content
- Makes **no guarantees** about delivery or the order of messages.
- Apps **do not have to be running** to receive notifications
  - the system will wake up the application via an Intent broadcast
- only passes **raw data** received to the application
- Requirements:
  - devices running Android 2.2 or above
  - have the **Market application** installed (Play Services)
  - a logged in **Google account**
- launched in 2010, officially **deprecated** as of June 26, 2012!
  - existing apps are **still working**, though

Google Cloud Messaging (GCM)

- successor of G2DM
- main differences:
  - to use the GCM service, you need to **obtain a Simple API Key** from the Google APIs console page
  - in C2DM, the Sender ID is an email address. In GCM, the **Sender ID** is a project number (acquired from the API console)
  - GCM HTTP requests **support JSON format** in addition to plain text
  - In GCM you can send the same message to multiple devices simultaneously (**multicast messaging**)
  - **Multiple parties** can send messages to the same app with one common registration ID
  - apps can send expiring invitation events with a **time-to-live** value between 0 and 4 weeks
    - GCM will store the messages until they expire
  - "**messages with payload**" to deliver messages of up to 4 Kb
  - GCM will store up to 100 messages
  - GCM provides **client and server helper libraries**

GCM architecture (1)

- GCM components
  - **Mobile Device**

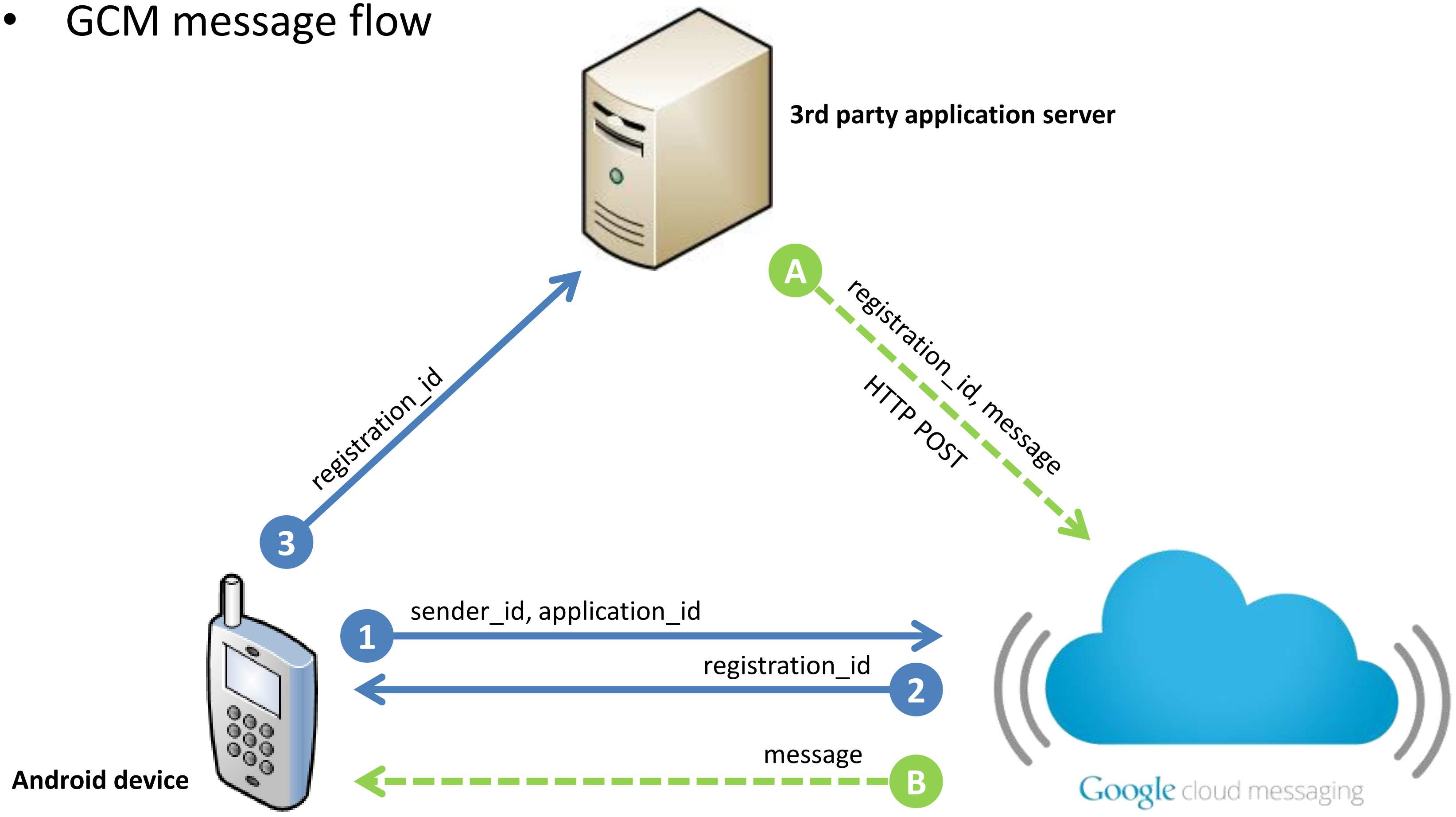
    - running an Android application that uses GCM
    - must be a 2.2 Android device that has Google Play Store installed
    - must have at least one logged in Google account
  - **3rd-party Application Server**

    - a server set up by an app developer as part of implementing GCM
    - sends data to an Android application on the device via GCM
  - **Coogle Cloud Messaging Servers**

    - the Google servers involved in taking messages from the 3rd-party application server and sending them to the device

GCM architecture (2)

- GCM message flow



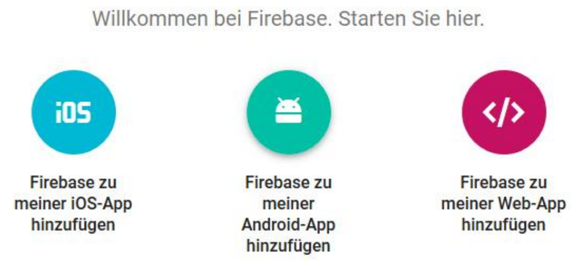
Firebase Cloud Messaging (FCM)

- About to be the successor of GCM
- **Free, reliable** cross-platform messaging
- Part of the Firebase Web Application Platform
- Key Capabilities:
  - Send **notification** or **data** messages
  - **Versatile** message **targeting**
  - **Two way** communication
- Migration from GCM implementations to FCM is necessary:

  `https://developers.google.com/cloud-messaging/android/android-migrate-fcm`
  - Android Manifest, listener classes, and server endpoints need to be adjusted

Using FCM with Java and Android (1)

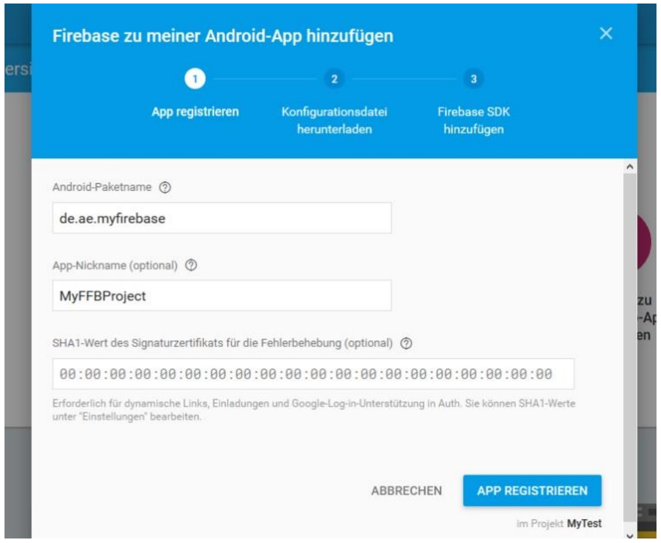
- Create a **new Firebase project** within the Google Firebase console
  - https://console.firebase.google.com/
  - Enter **project name** and **region**


Willkommen bei Firebase. Starten Sie hier.
iOS
Firebase zu meiner iOS-App hinzufügen
Firebase zu meiner Android-App hinzufügen
Firebase zu meiner Web-App hinzufügen

- Select **add Firebase Project to Android**

- Add your projects **package name** as well as a **nickname** (optional)

- **google-services.json** is generated and downloadable after clicking **register app**


Firebase zu meiner Android-App hinzufügen

1 App registrieren
2 Konfigurationsdatei herunterladen
3 Firebase SDK hinzufügen

Android-Paketname: de.ae.myfirebase

App-Nickname (optional): MyFFBProject

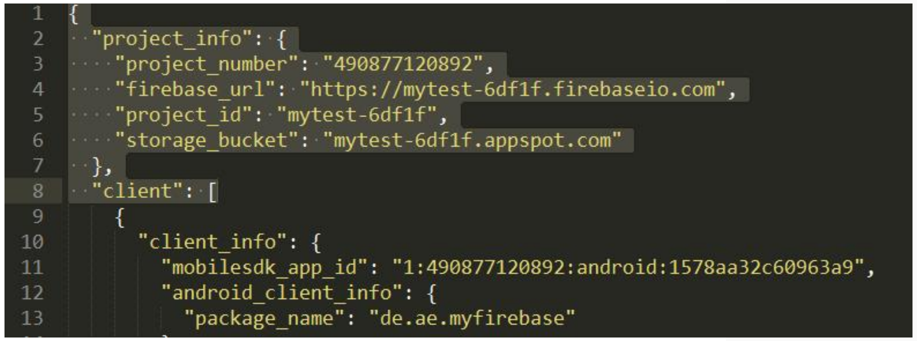
SHA1-Wert des Signaturzertifikats für die Fehlerbehebung (optional): 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00

Erforderlich für dynamische Links, Einladungen und Google-Log-in-Unterstützung in Auth. Sie können SHA1-Werte unter "Einstellungen" bearbeiten.

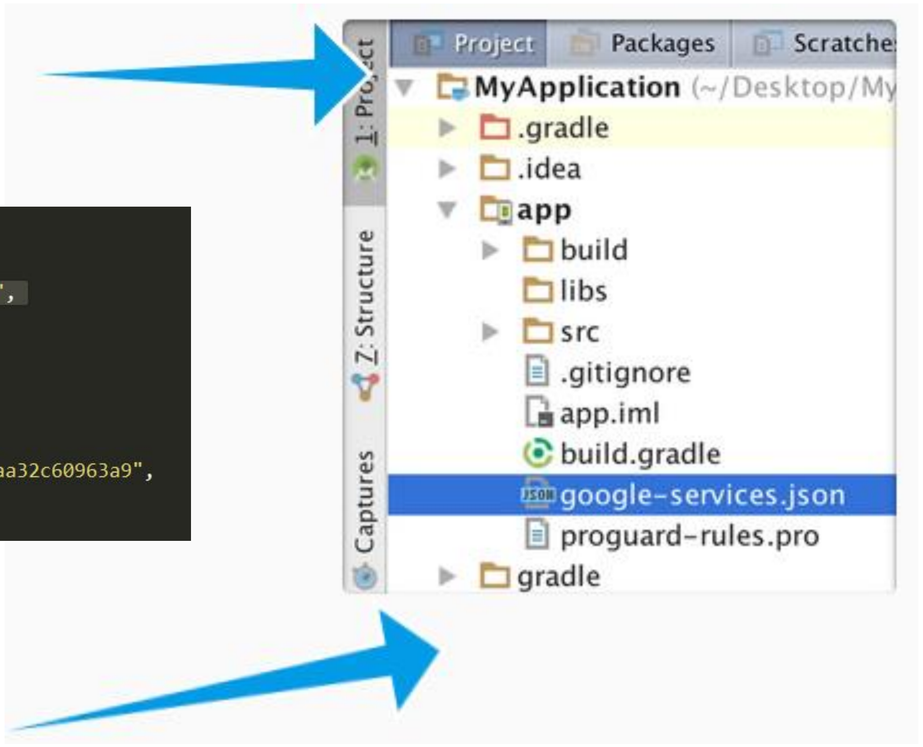
Using FCM with Java and Android (2)

1. Download **google-services.json**

2. Switch to Android Studio **project view**



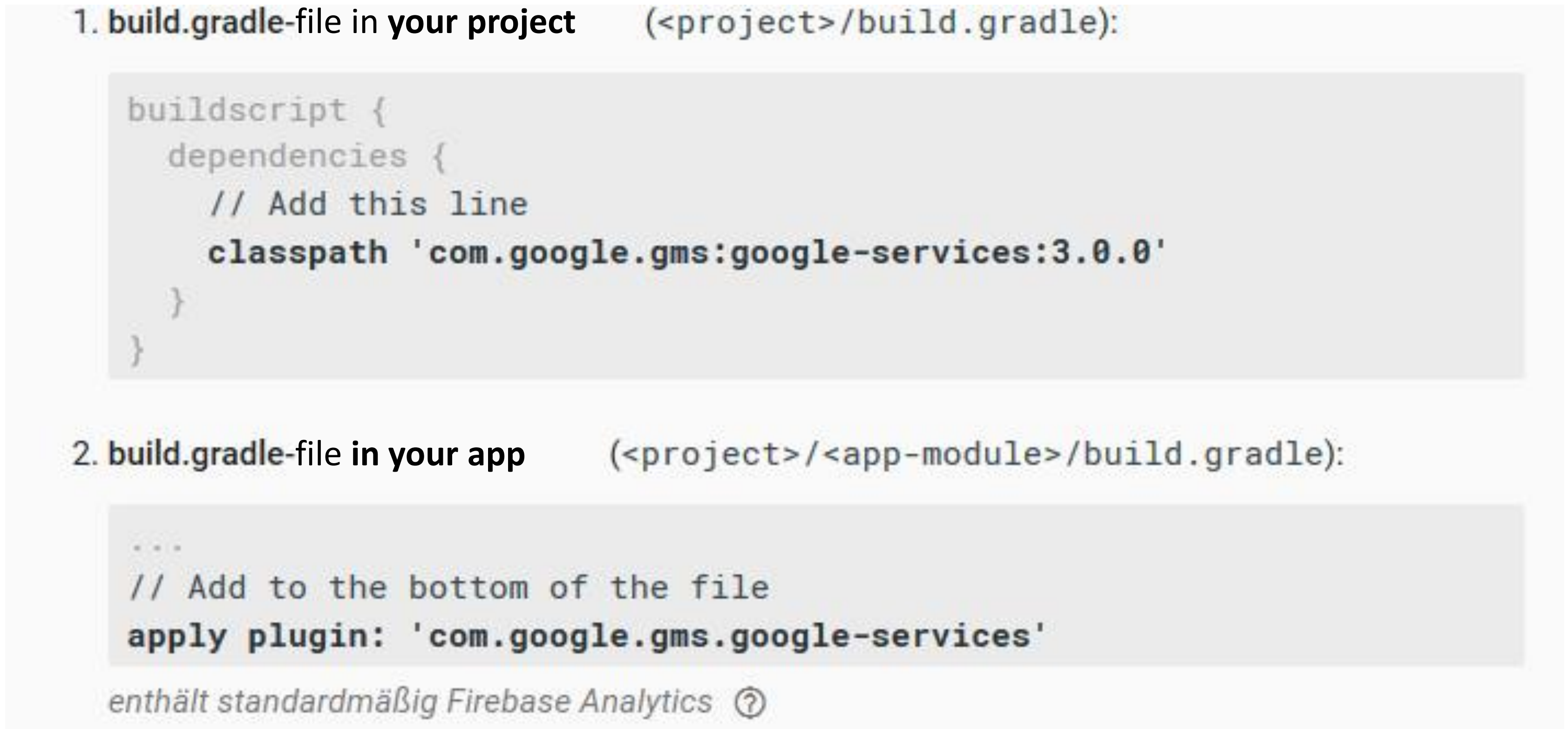
```
{
  "project_info": {
    "project_number": "490877120892",
    "firebase_url": "https://mytest-6df1f.firebaseio.com",
    "project_id": "mytest-6df1f",
    "storage_bucket": "mytest-6df1f.appspot.com"
  },
  "client": [
    {
      "client_info": {
        "mobilesdk_app_id": "1:490877120892:android:1578aa32c60963a9",
        "android_client_info": {
          "package_name": "de.ae.myfirebase"
```



```
1: Project    Project   Packages   Scratche
              ▼ MyApplication (~/Desktop/My
7: Structure    ▶ .gradle
                ▶ .idea
Captures        ▼ app
                  ▶ build
                    libs
                  ▶ src
                    .gitignore
                    app.iml
                    build.gradle
                    google-services.json
                    proguard-rules.pro
                ▶ gradle
```

3. Move google-services.json into the **root directory** of your Android App

Using FCM with Java and Android (3)



1. **build.gradle**-file in **your project** (<project>/build.gradle):

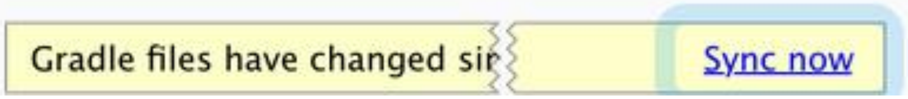
```
buildscript {
  dependencies {
    // Add this line
    classpath 'com.google.gms:google-services:3.0.0'
  }
}
```

2. **build.gradle**-file **in your app** (<project>/<app-module>/build.gradle):

```
...
// Add to the bottom of the file
apply plugin: 'com.google.gms.google-services'
```

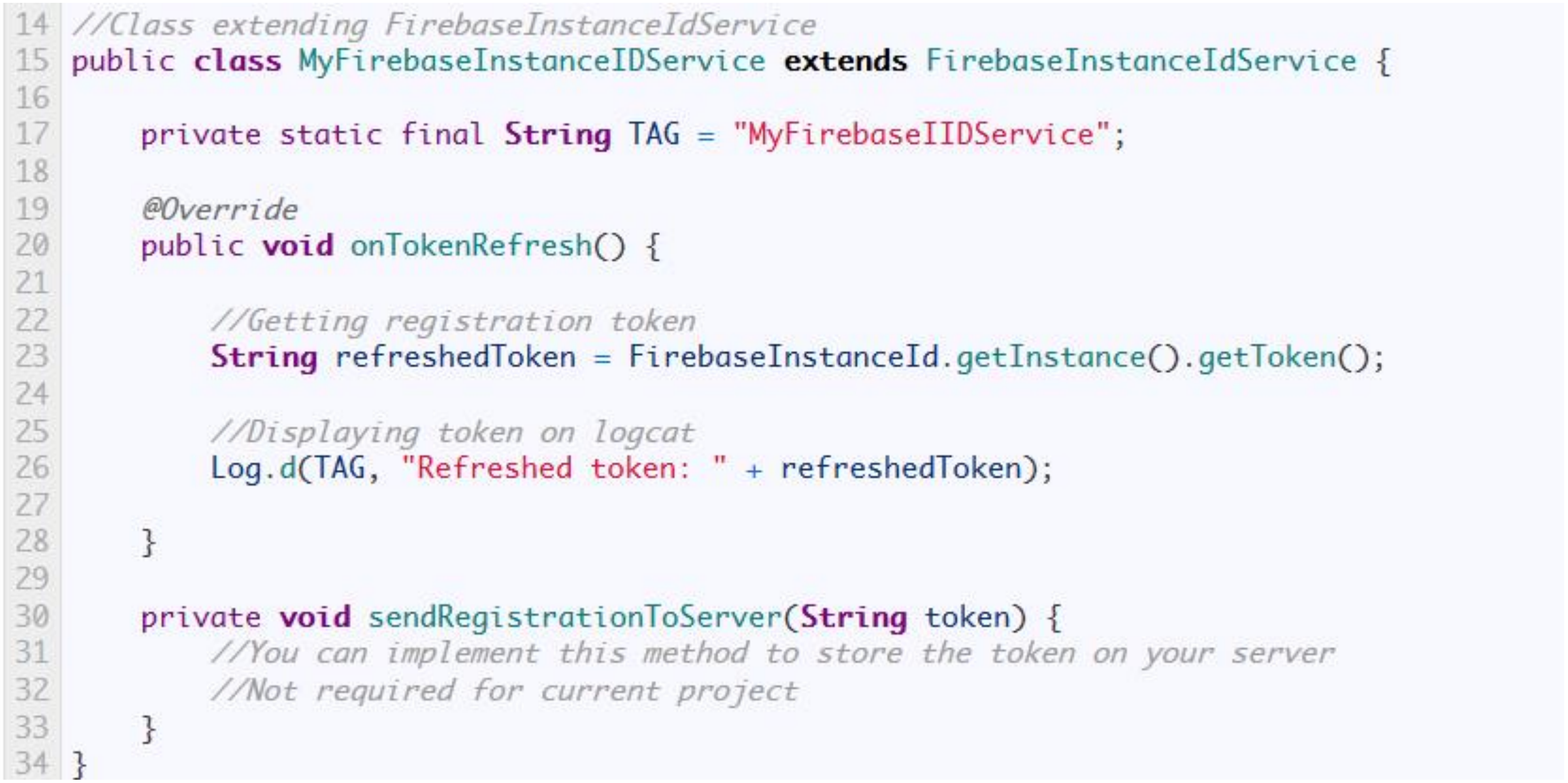
enthält standardmäßig Firebase Analytics

3. Click **Sync now** to apply changes and to make messaging services available within your project



Using FCM with Java and Android (4)

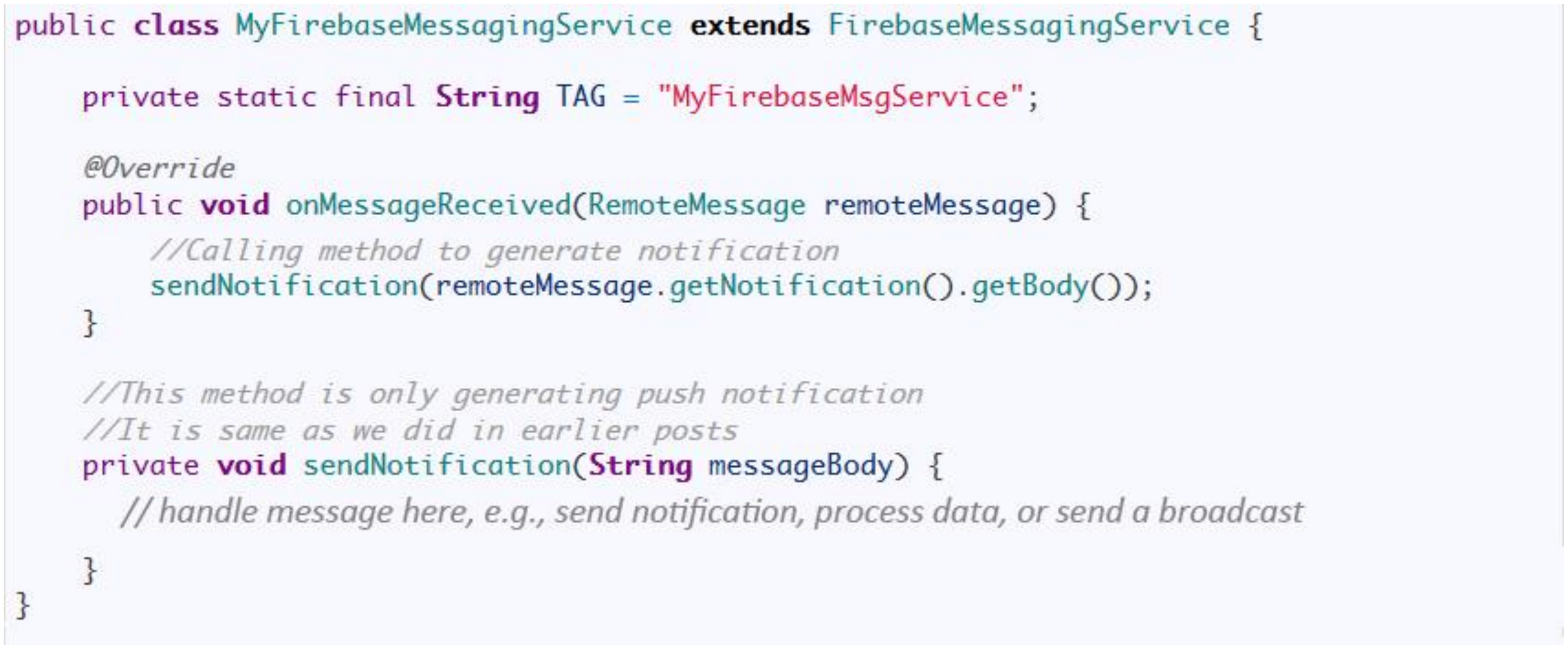
Create **FirebaseInstanceIdService** for token management:



```java
//Class extending FirebaseInstanceIdService
public class MyFirebaseInstanceIDService extends FirebaseInstanceIdService {

    private static final String TAG = "MyFirebaseIIDService";

    @Override
    public void onTokenRefresh() {

        //Getting registration token
        String refreshedToken = FirebaseInstanceId.getInstance().getToken();

        //Displaying token on logcat
        Log.d(TAG, "Refreshed token: " + refreshedToken);

    }

    private void sendRegistrationToServer(String token) {
        //You can implement this method to store the token on your server
        //Not required for current project
    }
}
```

Using FCM with Java and Android (5)

- Create **MyFirebaseMessagingService** to handle incoming messages



```java
public class MyFirebaseMessagingService extends FirebaseMessagingService {

    private static final String TAG = "MyFirebaseMsgService";

    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        //Calling method to generate notification
        sendNotification(remoteMessage.getNotification().getBody());
    }

    //This method is only generating push notification
    //It is same as we did in earlier posts
    private void sendNotification(String messageBody) {
      // handle message here, e.g., send notification, process data, or send a broadcast

    }
}
```

- Add your two new service classes to **AndroidManifest.xml**

Next Steps and further information

- Find tutorials and detailed implementation examples at
  - https://console.firebase.google.com/
- Implement a cloud messaging service within the scope of exercise 2, **REST and Push**