Inhaltsverzeichnis

1. Implementierung von Push-Benachrichtigungen in einer MAUI-App unter Verwendung von Firebase Cloud Messaging 3

1.1 Einführung 3

1.2 Gesamtsystem-Architektur der Push-Benachrichtigungen mit Firebase Cloud Messaging 4

2. .NET MAUI 5

2.1 Einführung 5

2.2 Projektbestandteile: 6

2.3 Schritt-für-Schritt-Anleitung (GitHub-Repository) 6

3. Firebase 15

3.1 Einführung 16

3.2 Vorteile von Firebase 16

4. Firebase Cloud Messaging (FCM) 16

4.1 Einführung 16

4.2 Hauptmerkmale von FCM 17

4.3 Vorteile von FCM 17

4.4 FCM-Architektur 18

4.5 Lebenszyklusfluss und Funktionsweise 18

4.5.1 Registrieren Sie Geräte, um Nachrichten von FCM zu empfangen. 18

4.5.2 Senden und empfangen Sie Downstream-Nachrichten. 19

5. Benutzerinstallation und Interaktion mit FCM 20

5.1 Installation der Anwendung 20

5.1.1 Download und Installation 20

5.1.2 Erste App-Initialisierung 21

5.2 Benutzeranmeldung und Interaktion mit Firebase 21

5.2.1 Anmeldung 21

5.2.2 Interaktion mit Firebase-Diensten 21

5.3 Interaktion mit FCM 21

5.3.1 FCM-SDK Integration 22

5.3.2 Initialisierung des Dienstes 22

5.3.3 Registrierungstoken Handhabung 22

5.3.4 Senden des Tokens an den Server 22

5.4 FCM-Nachrichten 23

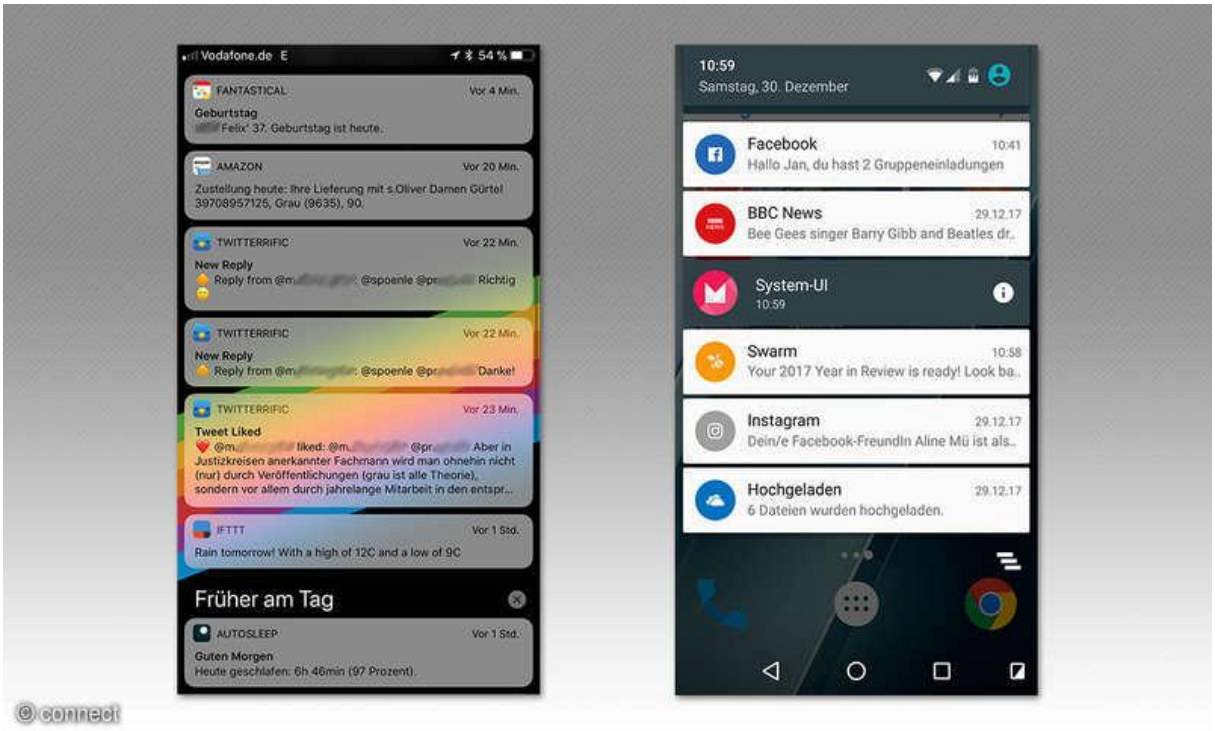
5.5 FCM-Registrierungstoken-Verwaltung 24
5.5.1 Veraltete und abgelaufene Registrierungstoken 24
5.5.2 Grundlegende Best Practices 24
5.5.3 Registrierungstoken abrufen und speichern 24
5.5.4 Aktualität der Tokens sicherstellen und veraltete Tokens entfernen 25
5.5.5 Tokens regelmäßig aktualisieren 25
6. Serverumgebung und FCM 25
6.1 Anforderungen an die vertrauenswürdige Serverumgebung 25
6.2 Firebase Admin SDK für FCM 26
6.3 Voraussetzungen 26
7. Integration des Firebase Admin SDK in eine .NET-Konsolen App 27
7.1 Einführung 27
7.2 Schritt-für-Schritt-Anleitung 27
8. Literaturverzeichnis 30

1. Implementierung von Push-Benachrichtigungen in einer MAUI-App unter Verwendung von Firebase Cloud Messaging

1.1 Einführung

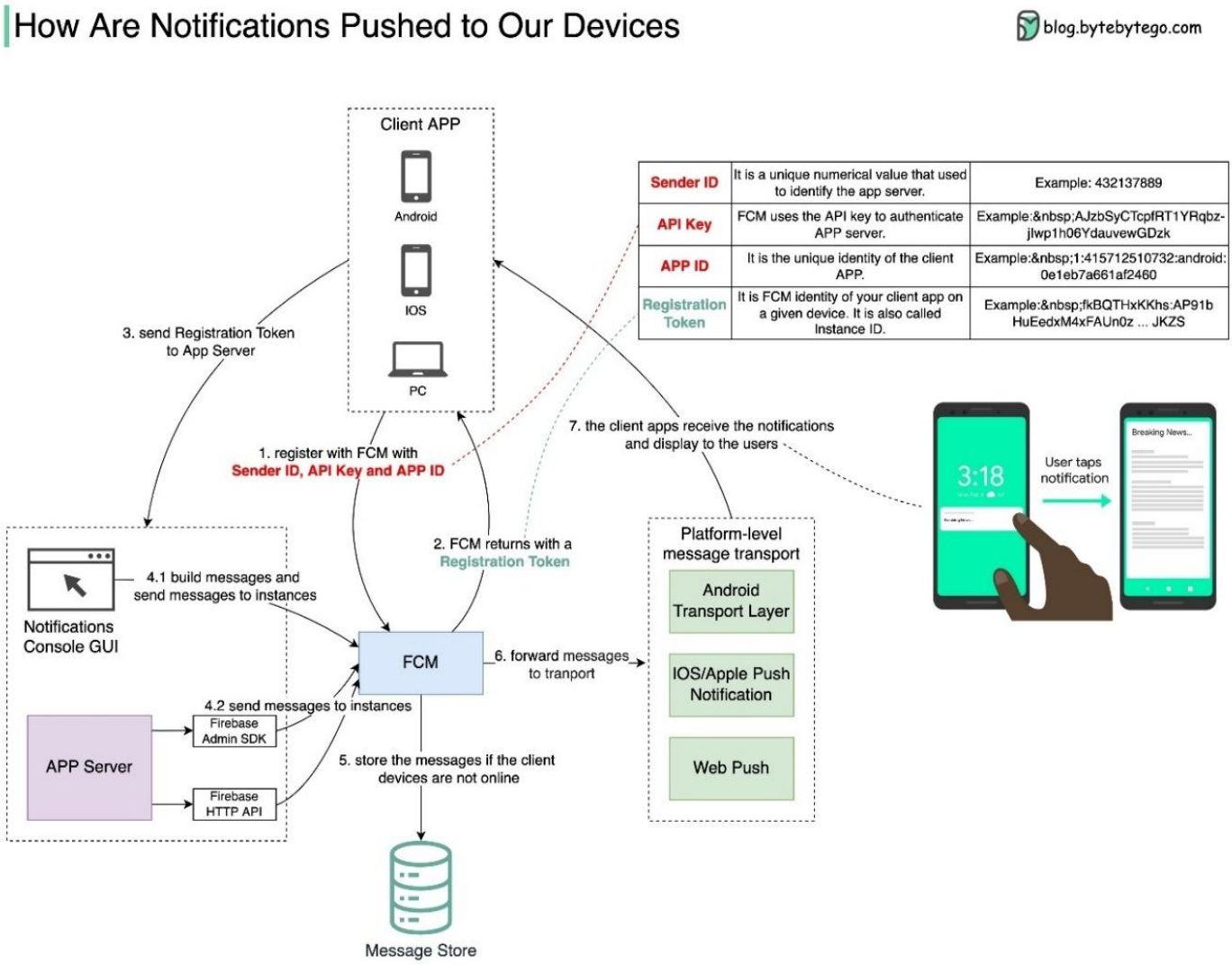
Eine Push-Benachrichtigung ist eine kurze Nachricht, die auf dem Desktop, Mobilgerät oder in der Benachrichtigungszentrale erscheint. Sie kann Text und Medien enthalten und motiviert Nutzer zu bestimmten Aktionen. Es gibt webbasierte und app-basierte Push-Benachrichtigungen. Unternehmen nutzen sie zur Kundenbindung, Umsatzsteigerung und zur Übermittlung wichtiger Informationen. Auch als Sicherheitsmechanismus, etwa zur Authentifizierung, sind sie nützlich.

Push-Benachrichtigungen bieten höhere Öffnungsraten als E-Mails, ermöglichen Echtzeit-Kommunikation und können personalisiert werden. Der Erfolg hängt von gutem Timing, Personalisierung und Segmentierung ab. Sie werden in vielen Branchen eingesetzt, besonders in Bereichen wie Finanzen, Gesundheit und Verkehr. Automatisierung und Anpassungsmöglichkeiten verbessern die Benutzererfahrung und erhöhen die Interaktion.



1.2 Gesamtsystem-Architektur der Push-Benachrichtigungen mit Firebase Cloud Messaging

Das Bild zeigt die Gesamtsystem-Architektur der Push-Benachrichtigungen mit Firebase Cloud Messaging (FCM) und verdeutlicht die Kommunikationsflüsse vom Backend-Server über FCM zur Client-App auf dem Benutzergerät.

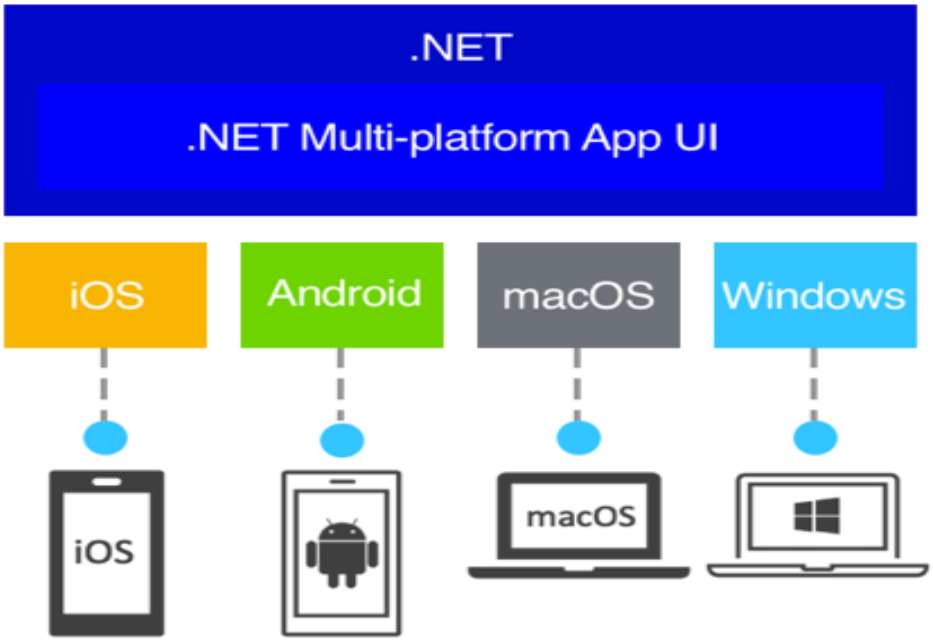


2. .NET MAUI

2.1 Einführung

.NET MAUI (Multi-Platform App UI) ist ein plattformübergreifendes Framework zum Erstellen nativer Mobil- und Desktop-Apps mit C# und XAML. Damit können Apps entwickelt werden, die auf Android, iOS, macOS und Windows ausgeführt werden können. Als Open-Source-Weiterentwicklung von Xamarin.Forms erweitert um Desktop-Szenarien ermöglicht es die Entwicklung plattformübergreifender Apps mit einem einzigen Projekt, wobei bei Bedarf plattformspezifischer Code hinzugefügt werden kann. Das Ziel ist, möglichst viel App-Logik und UI in einer Codebasis zu bündeln.

.NET MAUI vereint Android-, iOS-, macOS- und Windows-APIs in einer einzigen API, die eine Write-Once-Run-Anywhere-Entwicklung ermöglicht und gleichzeitig umfassenden Zugriff auf alle Aspekte der jeweiligen nativen Plattform bietet.



2.2 Projektbestandteile:

MAUI-App

- **Frameworks**: .NET MAUI-App
- **Hauptfunktionen**: Registrierung für Push-Benachrichtigungen, Erzeugung des Zugriffstokens

Firebase

- **Dienste**: Firebase Cloud Messaging
- **Hauptfunktionen**: Vermittlung von Benachrichtigungen

Server

- **Frameworks**: .NET-Konsolen App
- **Hauptfunktionen**: Verwaltung und Versendung von Push-Benachrichtigungen

2.3 Schritt-für-Schritt-Anleitung (GitHub-Repository)

1) Projekt erstellen

Erstellen Sie ein neues ***.NET MAUI App***-Projekt in Ihrer Entwicklungsumgebung.

2) NuGet-Pakete installieren

Projekt Manifest anpassen und eventuell die Pakete noch aktualisieren.

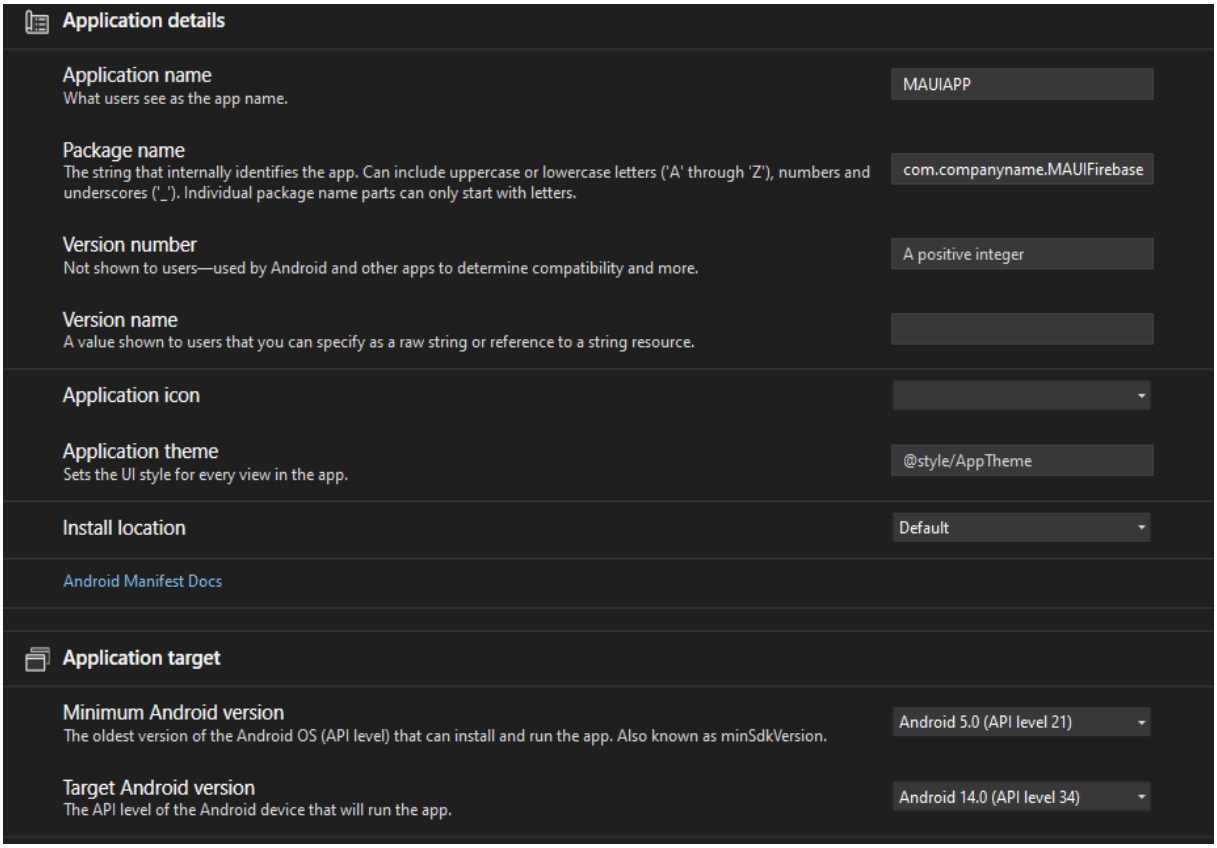
```
<Project Sdk="Microsoft.NET.Sdk">

	<PropertyGroup>
		<TargetFrameworks>net8.0-android</TargetFrameworks>
		<OutputType>Exe</OutputType>
		<RootNamespace>MAUIFirebaseNotification</RootNamespace>
		<UseMaui>true</UseMaui>
		<SingleProject>true</SingleProject>
		<ImplicitUsings>enable</ImplicitUsings>
		<Nullable>enable</Nullable>

		<!-- Display name -->
		<ApplicationTitle>MAUIFirebaseNotification</ApplicationTitle>

		<!-- App Identifier -->

	<ApplicationId>com.companyname.mauifirebasenotification</ApplicationId>

		<!-- Versions -->
		<ApplicationDisplayVersion>1.0</ApplicationDisplayVersion>
		<ApplicationVersion>1</ApplicationVersion>

		<SupportedOSPlatformVersion
Condition="$([MSBuild]::GetTargetPlatformIdentifier('$(TargetFramework)')) ==
'android'">21.0</SupportedOSPlatformVersion>
	</PropertyGroup>
```

```
	<ItemGroup>
		<!-- App Icon -->
		<MauiIcon Include="Resources\AppIcon\appicon.svg"
ForegroundFile="Resources\AppIcon\appiconfg.svg" Color="#512BD4" />

		<!-- Splash Screen -->
		<MauiSplashScreen Include="Resources\Splash\splash.svg"
Color="#512BD4" BaseSize="128,128" />

		<!-- Images -->
		<MauiImage Include="Resources\Images\*" />
		<MauiImage Update="Resources\Images\dotnet_bot.png" Resize="True"
BaseSize="300,185" />

		<!-- Custom Fonts -->
		<MauiFont Include="Resources\Fonts\*" />

		<!-- Raw Assets (also remove the "Resources\Raw" prefix) -->
		<MauiAsset Include="Resources\Raw\**"
LogicalName="%(RecursiveDir)%(Filename)%(Extension)" />
	</ItemGroup>

	<ItemGroup>
		<None Remove="google-services.json" />
		<None Remove="Platforms\Android\Resources\values\strings.xml" />
	</ItemGroup>

	<ItemGroup Condition="'$(TargetFramework)' == 'net8.0-android'">
		<GoogleServicesJson Include="google-services.json" />
	</ItemGroup>

	<ItemGroup>
		<PackageReference Include="Microsoft.Maui.Controls" Version="8.0.21"
/>
		<PackageReference Include="Microsoft.Maui.Controls.Compatibility"
Version="8.0.21" />
		<PackageReference Include="Microsoft.Extensions.Logging.Debug"
Version="8.0.0" />
		<PackageReference Include="Plugin.Firebase.Crashlytics"
Version="2.0.3" />
		<PackageReference Include="Plugin.Firebase" Version="2.0.14" />
	</ItemGroup>

	<ItemGroup Condition="'$(TargetFramework)' == 'net8.0-android'">
		<PackageReference Include="Xamarin.AndroidX.Preference">
			<Version>1.2.1.4</Version>
		</PackageReference>
	</ItemGroup>

	<ProjectExtensions><VisualStudio><UserProperties
XamarinHotReloadDebuggerTimeoutExceptionMAUIPushNotifictionHideInfoBar="True"
/></VisualStudio></ProjectExtensions>

</Project>
```

3) *AndroidManifest.xml* anpassen

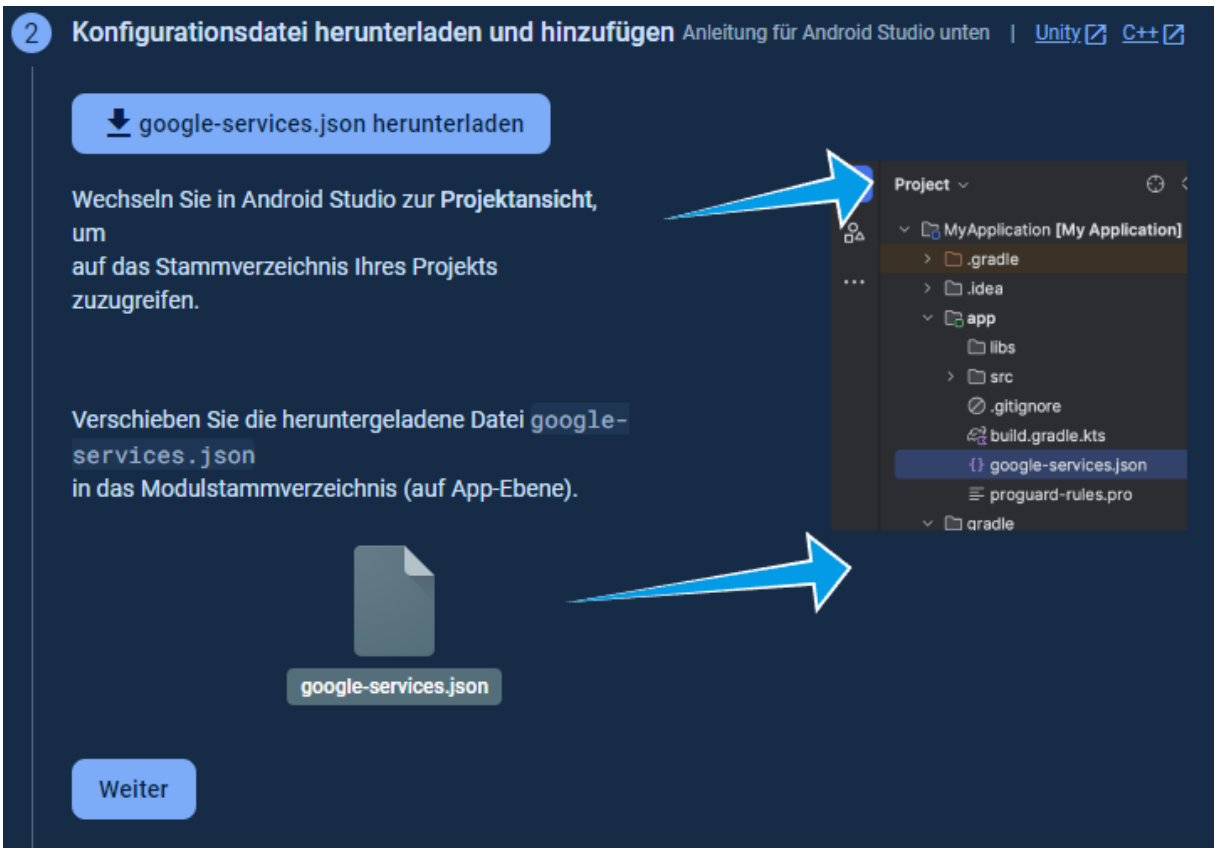
Passen Sie die AndroidManifest.xml an, um die erforderlichen Berechtigungen und Empfänger hinzuzufügen:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.companyname.MAUIFirebaseNotification">
	<application android:allowBackup="true" android:supportsRtl="true" android:label="MAUIAPP"
android:icon="@mipmap/appicon">
		<receiver android:name="com.google.firebase.iid.FirebaseInstanceIdInternalReceiver"
android:exported="false" />
		<receiver android:name="com.google.firebase.iid.FirebaseInstanceIdReceiver"
android:exported="true" android:permission="com.google.android.c2dm.permission.SEND">
			<intent-filter>
				<action android:name="com.google.android.c2dm.intent.RECEIVE" />
				<action
android:name="com.google.android.c2dm.intent.REGISTRATION" />
				<category android:name="${applicationId}" />
			</intent-filter>
		</receiver>
	</application>
	<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
	<uses-permission android:name="android.permission.INTERNET" />
	<uses-permission android:name="android.permission.WAKE_LOCK" />
	<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
	<uses-sdk android:minSdkVersion="21" android:targetSdkVersion="34" />

</manifest>
```



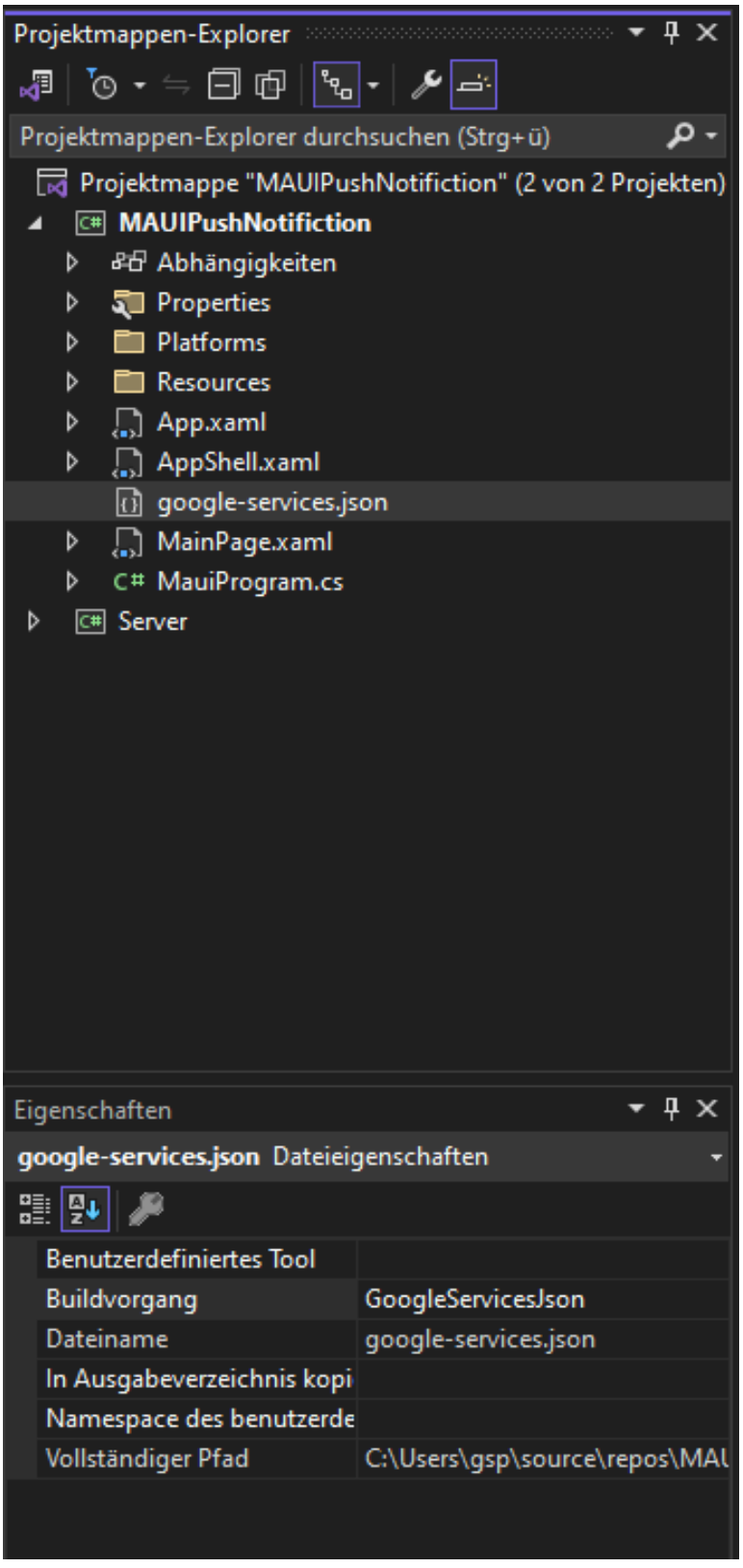
4) Firebase-Projekt einrichten

Erstellen Sie ein Projekt in der Firebase-Konsole und fügen Sie Ihre App hinzu mit ***Paketname*** (aus Android Manifest) und ***Nickname*** . Laden Sie die google-services.json Datei herunter und kopieren Sie sie in das Resources Verzeichnis Ihres Android-Projekts.



5) Google-Service einfügen

In Android Projekt die Datei ***google-Service.json*** im Ordner Resources kopiert und in Eigenschaften ***Buildvorgang - Google ServicesJson:***



6) MAUI-Konfigurationsdateien anpassen

Fügen Sie die erforderlichen Klassen und Konfigurationsdateien zu Ihrem Projekt hinzu.

- ***MauiProgramm.cs***

```
using Microsoft.Maui.LifecycleEvents;
using Plugin.Firebase.Auth;
using Plugin.Firebase.Bundled.Shared;
using Microsoft.Extensions.Logging;
using Plugin.Firebase.Crashlytics;
using MAUIPushNotifiction;



#if IOS
using Plugin.Firebase.Bundled.Platforms.iOS;
#else
using Plugin.Firebase.Bundled.Platforms.Android;
#endif

namespace MAUIFirebaseNotification
{
  public static class MauiProgram
  {
    public static MauiApp CreateMauiApp()
    {
      var builder = MauiApp.CreateBuilder();
      builder
        .UseMauiApp<App>()
        .ConfigureFonts(fonts =>
        {
          fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
          fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
        });

#if DEBUG
      builder.Logging.AddDebug();
#endif

      return builder.Build();
    }

    private static MauiAppBuilder RegisterFirebaseServices(this MauiAppBuilder builder)
    {
      builder.ConfigureLifecycleEvents(events =>
      {
#if IOS
      events.AddiOS(iOS => iOS.FinishedLaunching((app, launchOptions) => {
        CrossFirebase.Initialize(CreateCrossFirebaseSettings());
        return false;
      }));
#else
        events.AddAndroid(android => android.OnCreate((activity, _) =>
          CrossFirebase.Initialize(activity, CreateCrossFirebaseSettings())));
        CrossFirebaseCrashlytics.Current.SetCrashlyticsCollectionEnabled(true);
#endif
      });

      builder.Services.AddSingleton(_ => CrossFirebaseAuth.Current);
      return builder;
    }
```

```
    private static CrossFirebaseSettings CreateCrossFirebaseSettings()
    {
      return new CrossFirebaseSettings(isAuthEnabled: true,
      isCloudMessagingEnabled: true, isAnalyticsEnabled: true);
    }
  }
}
```

- ***Platforms > Android > Resources > Values***, neue Klasse ***strings.xml*** einfügen (Einstellungen auf AndroidResources setzen)

```
<?xml version="1.0" encoding="utf-8" ?>
<resources>
        <string name="com.google.firebase.crashlytics.mapping_file_id">none</string>
</resources
```

- ***Platforms*** > ***Android*** > ***MainActivity.cs***

```
using Android.App;
using Android.Content;
using Android.Content.PM;
using Android.OS;
using Plugin.Firebase.CloudMessaging;

namespace MAUIFirebaseNotification
{
  [Activity(Theme = "@style/Maui.SplashTheme", MainLauncher = true, ConfigurationChanges =
ConfigChanges.ScreenSize | ConfigChanges.Orientation | ConfigChanges.UiMode |
ConfigChanges.ScreenLayout | ConfigChanges.SmallestScreenSize | ConfigChanges.Density)]
  public class MainActivity : MauiAppCompatActivity
  {
    protected override void OnCreate(Bundle savedInstanceState)
    {
      base.OnCreate(savedInstanceState);
      HandleIntent(Intent);
      CreateNotificationChannelIfNeeded();
    }

    protected override void OnNewIntent(Intent intent)
    {
      base.OnNewIntent(intent);
      HandleIntent(intent);
    }

    private static void HandleIntent(Intent intent)
    {
      FirebaseCloudMessagingImplementation.OnNewIntent(intent);
    }

    private void CreateNotificationChannelIfNeeded()
    {
      if (Build.VERSION.SdkInt >= BuildVersionCodes.O)
      {
```

```
            CreateNotificationChannel();
        }
    }

    private void CreateNotificationChannel()
    {
        var channelId = $"{PackageName}.general";
        var notificationManager = (NotificationManager)GetSystemService(NotificationService);
        var channel = new NotificationChannel(channelId, "General", NotificationImportance.Default);
        notificationManager.CreateNotificationChannel(channel);
        FirebaseCloudMessagingImplementation.ChannelId = channelId;

    }
  }
}
```

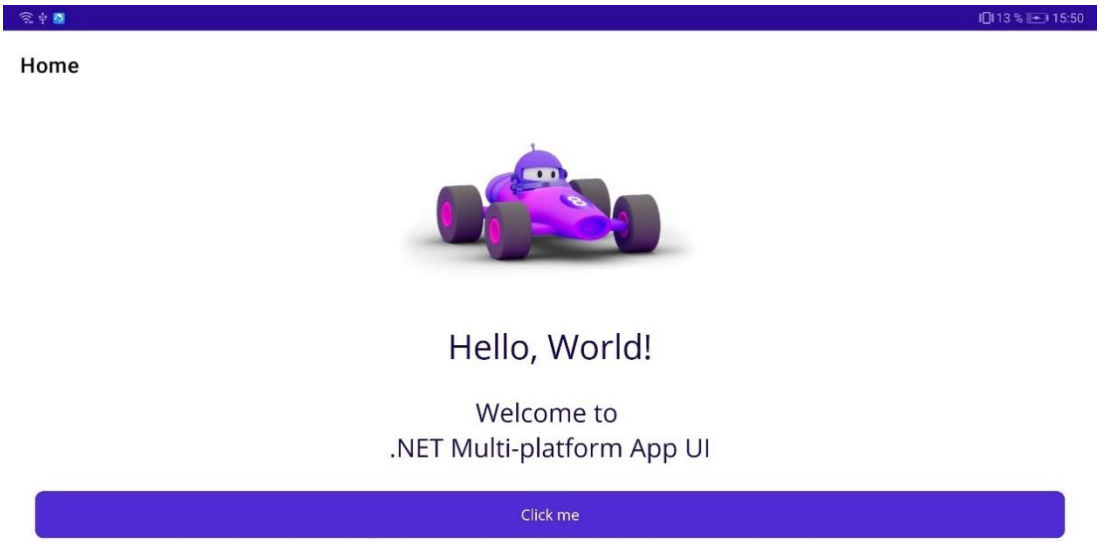
- ***MainPage.xaml.cs***

```
using Microsoft.Maui.Controls;
using Plugin.Firebase.CloudMessaging;

namespace MAUIPushNotifiction
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }

        private async void OnCounterClicked(object sender, EventArgs e)
        {
            await CrossFirebaseCloudMessaging.Current.CheckIfValidAsync();
            var token = await
CrossFirebaseCloudMessaging.Current.GetTokenAsync();
            Console.WriteLine($"FCM token: {token}");
        }
    }
}
```

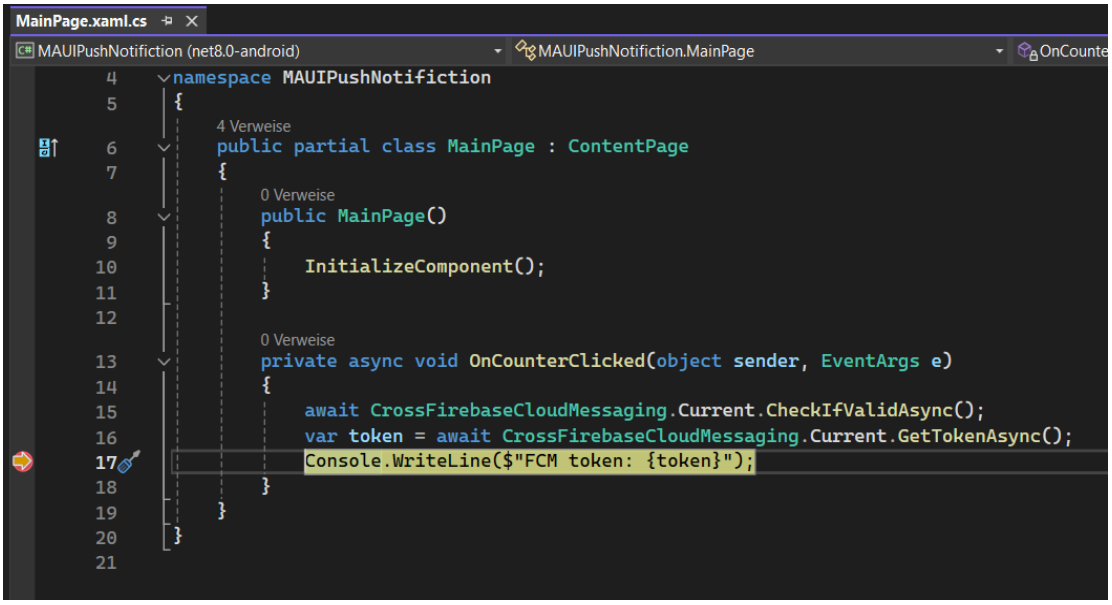
7) Token generieren und verwenden

In der Klasse ***MainPage.xaml.cs*** wird ein Haltepunkt in der Zeile ***Console.WriteLine($"FCM token: {token}")*** gesetzt.

Nach dem Start des Programms wird die .NET MAUI-Anwendung auf dem externen Gerät/Emulator installiert und anschließend das Token generiert. Um ein neues Token zu generieren, muss die Anwendung vom Gerät deinstalliert werden, andernfalls generiert die Anwendung immer dasselbe Token.



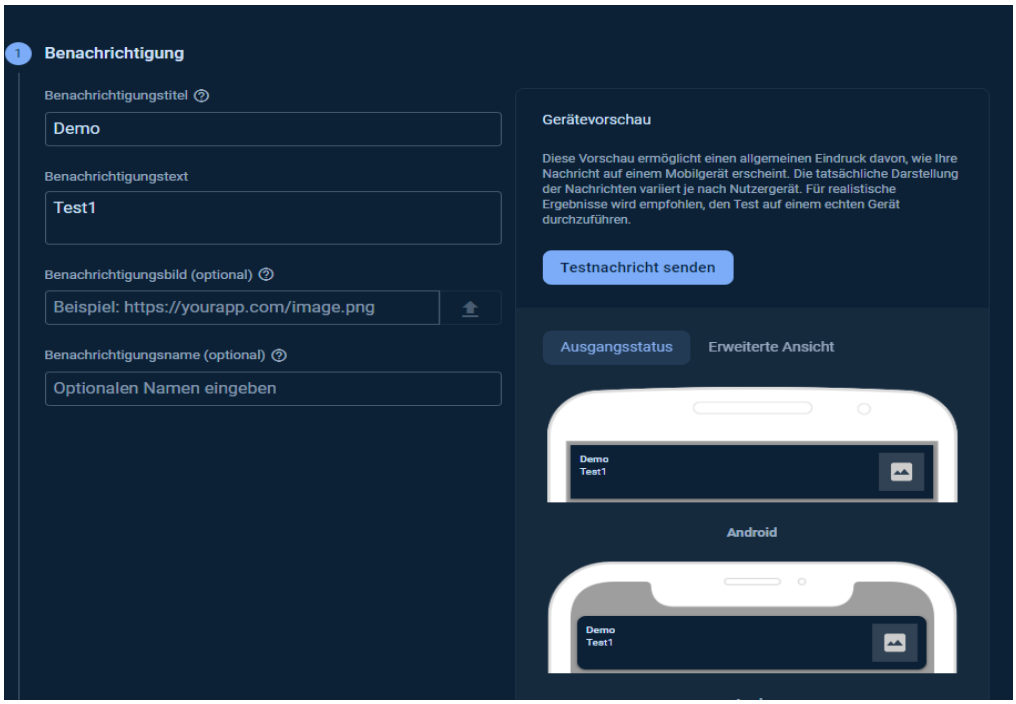
Das Token erscheint auf der Zeile, in der der Haltepunkt gesetzt wurde, mit gelbem Hintergrund. Von dort aus kann es kopiert und verwendet werden.



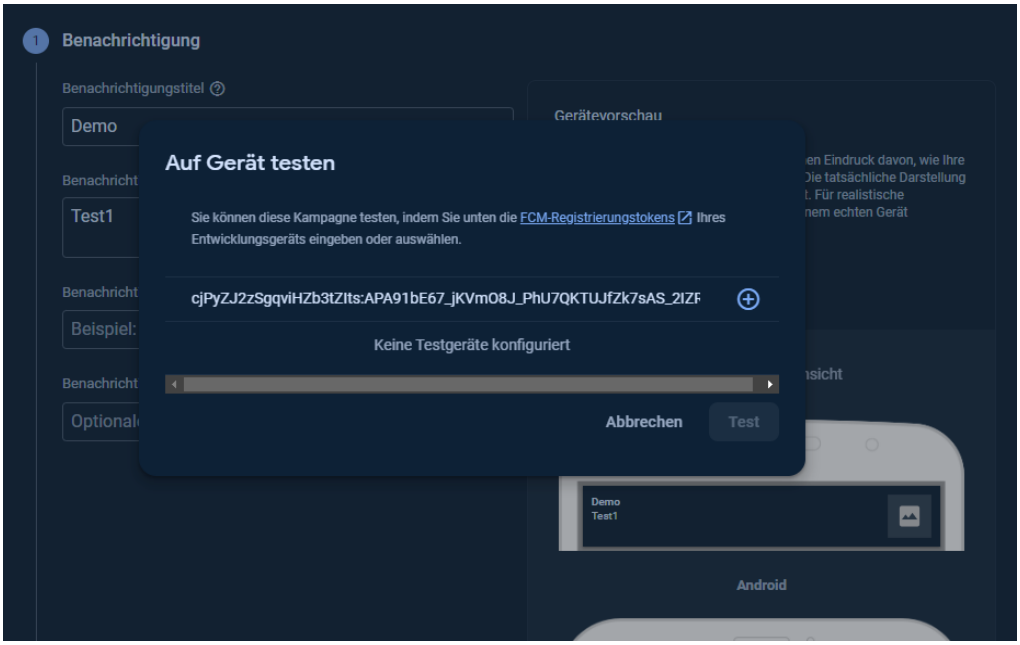
8) Nachricht senden über *Firebase Cloud Messaging*

➢ ***Firebase-Konsole > Cloud-Messaging > Neue Kampagne***

Titel und Text einfügen:



Token einfügen:



3. Firebase

3.1 Einführung

Firebase ist eine Plattform für mobile und Web-Anwendungen, die von Google entwickelt wurde. Sie bietet eine breite Palette an Tools und Diensten, die Entwicklern helfen, hochwertige Apps zu erstellen, zu verbessern und zu skalieren. Ursprünglich im Jahr 2011 als Echtzeit-Datenbankdienst gestartet, wurde Firebase 2014 von Google übernommen und seither kontinuierlich erweitert und verbessert.

3.2 Vorteile von Firebase

- **Einfache Integration:** SDKs für iOS, Android, Web und mehr ermöglichen eine einfache Integration in verschiedene Plattformen.
- **Skalierbarkeit:** Firebase-Dienste unterstützen sowohl kleine Projekte als auch große Anwendungen mit Millionen von Nutzern.
- **Echtzeit-Synchronisation:** Ideal für Anwendungen, die Echtzeit-Datenaktualisierungen benötigen.
- **Sicherheit:** Integrierte Sicherheits- und Authentifizierungsfunktionen schützen Daten und Benutzer.
- **Kostenfrei bis kostenpflichtig:** Viele Firebase-Dienste sind kostenlos nutzbar, mit kostenpflichtigen Optionen für erweiterte Funktionen und größere Nutzung.

Firebase ist eine umfassende Entwicklungsplattform, die Entwicklern dabei hilft, Anwendungen schneller zu erstellen, die Benutzerbindung zu verbessern und die Leistung zu optimieren.

4. Firebase Cloud Messaging (FCM)

4.1 Einführung

Firebase Cloud Messaging ist ein kostenloser Dienst von Google, der Entwicklern ermöglicht, Nachrichten und Benachrichtigungen an Benutzergeräte zu senden. FCM bietet eine zuverlässige und skalierbare Lösung für die Kommunikation mit Apps auf iOS-, Android- und Web-Plattformen.

4.2 Hauptmerkmale von FCM

- **Nachrichtenarten**: Senden Sie Benachrichtigungen, die den Benutzern angezeigt werden, oder Datennachrichten, die in Ihrem Anwendungscode verarbeitet werden.

- V**ielseitige Nachrichtenausgabe:** Nachrichten an einzelne Geräte, Gerätegruppen oder Geräte, die Themen abonniert haben.

- **Client-zu-Server-Kommunikation:** Senden Sie Bestätigungen, Chats und andere Nachrichten von Geräten zurück an Ihren Server über den zuverlässigen und batteriefreundlichen Verbindungskanal von FCM.

4.3 Vorteile von FCM

- **Kostenfrei:** FCM ist kostenlos nutzbar.

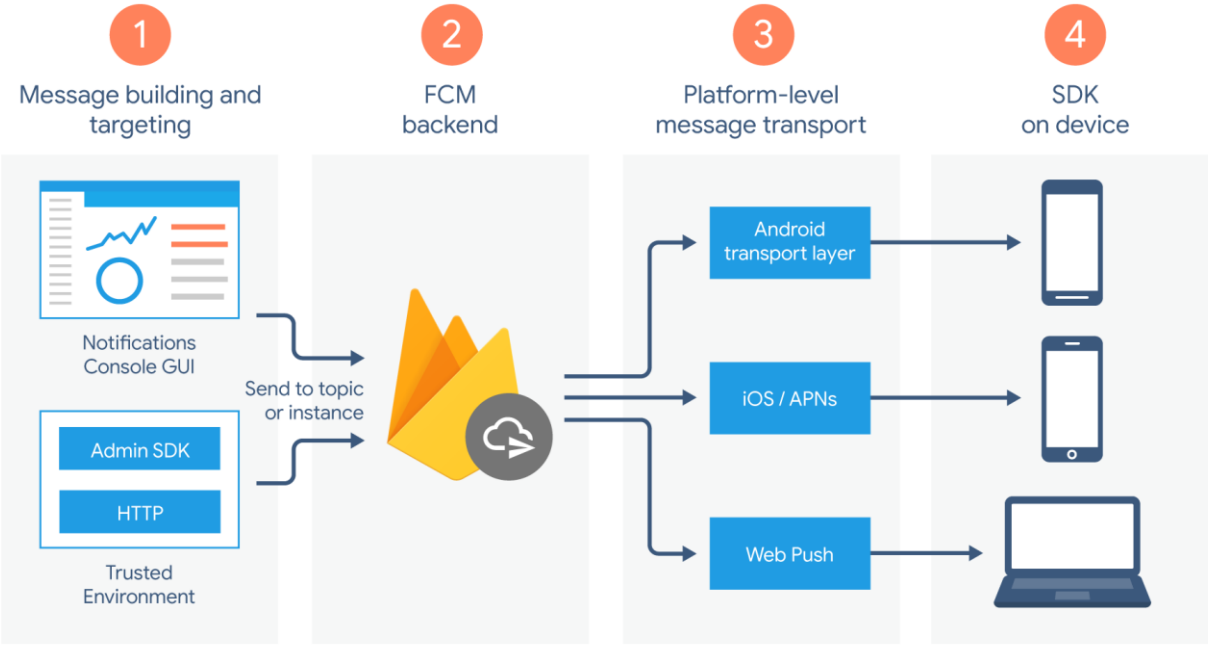
- **Einfache Integration:** Dank umfassender Dokumentation und SDKs für verschiedene Plattformen ist die Integration einfach.

- **Hohe Reichweite:** Unterstützt mehrere Plattformen und ermöglicht eine breite Nutzerbasis.

- **Personalisierung:** Nachrichten können personalisiert und gezielt versendet werden.

- **Echtzeit-Kommunikation:** Ideal für Anwendungen, die sofortige Benachrichtigungen erfordern, wie Nachrichten-Apps oder soziale Netzwerk

4.4 FCM-Architektur



Komponenten:

1. **Tools zum Verfassen von Nachrichten**
   - Firebase-Konsole und Admin SDKs zum Erstellen und Senden von Nachrichten.

2. **FCM-Backend**
   - Das von Google betriebene Backend verarbeitet und leitet Nachrichten an die entsprechenden Geräte weiter.

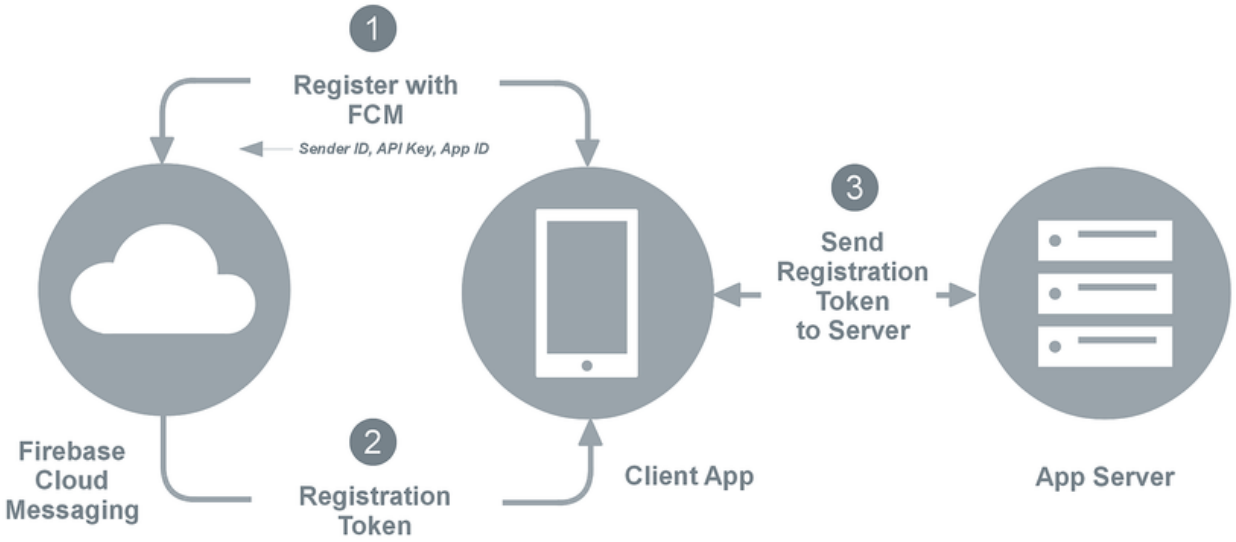
3. **Transportschicht auf Plattformebene**
   - Optimierte Netzwerkprotokolle, um Nachrichten effizient zu transportieren.

4. **FCM SDK auf dem Gerät des Benutzers**
   - Integrierte Bibliotheken in den Apps, die das Empfangen und Verarbeiten von Nachrichten ermöglichen.

4.5 Lebenszyklusfluss und Funktionsweise

4.5.1 Registrieren Sie Geräte, um Nachrichten von FCM zu empfangen.

Eine Client-App muss sich zuerst bei FCM registrieren, bevor das Messaging erfolgen kann. Die Client-App muss, die im folgenden Diagramm gezeigten Registrierungsschritte ausführen:



1. Die Client-App kontaktiert FCM, um ein Registrierungstoken zu erhalten, und übergibt die Absender-ID, den API-Schlüssel und die App-ID an FCM.

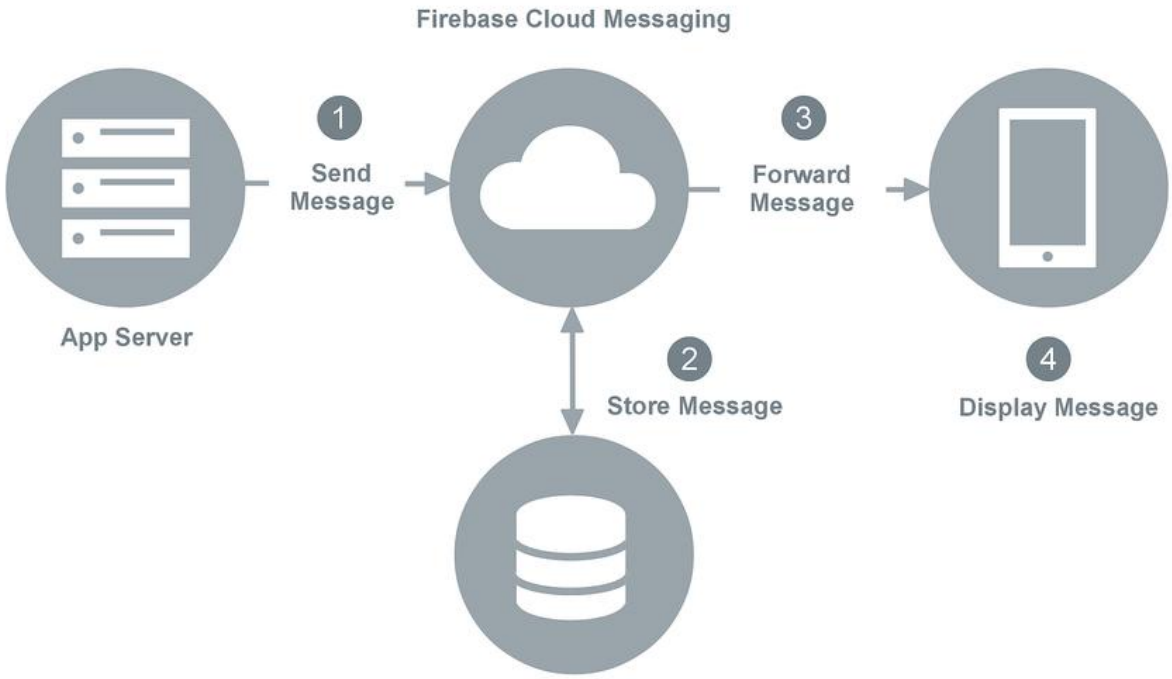
2. FCM gibt ein Registrierungstoken an die Client-App zurück.

3. Die Client-App leitet das Registrierungstoken (optional) an den App-Server weiter.

Der App-Server speichert das Registrierungstoken für die nachfolgende Kommunikation mit der Client-App zwischen. Der App-Server kann eine Bestätigung zurück an die Client-App senden, um anzugeben, dass das Registrierungstoken empfangen wurde. Nachdem dieser Handshake erfolgt ist, kann die Client-App Nachrichten vom App-Server empfangen (oder an diesen senden). Die Client-App erhält möglicherweise ein neues Registrierungstoken, wenn das alte Token kompromittiert ist.
Wenn die Client-App keine Nachrichten mehr vom App-Server empfangen möchte, kann sie eine Anforderung an den App-Server senden, um das Registrierungstoken zu löschen.
Wenn die Client-App von einem Gerät deinstalliert wird, erkennt FCM dies und benachrichtigt den App-Server automatisch, das Registrierungstoken zu löschen.

4.5.2 Senden und empfangen Sie Downstream-Nachrichten.

Das folgende Diagramm veranschaulicht, wie Firebase Cloud Messaging Downstream-Nachrichten speichert und weiterleitet:



Wenn der App-Server eine Downstreamnachricht an die Client-App sendet, werden die folgenden Schritte ausgeführt, wie im obigen Diagramm aufgeführt:

1. Der App-Server sendet die Nachricht an FCM.

2. Wenn das Clientgerät nicht verfügbar ist, speichert der FCM-Server die Nachricht zur späteren Übertragung in einer Warteschlange. Nachrichten werden maximal 4 Wochen lang im FCM-Speicher aufbewahrt (weitere Informationen finden Sie unter Festlegen der Lebensdauer einer Nachricht).

3. Wenn das Clientgerät verfügbar ist, leitet FCM die Nachricht an die Client-App auf diesem Gerät weiter.

4. Die Client-App empfängt die Nachricht von FCM, verarbeitet sie und zeigt sie dem Benutzer an. Wenn es sich bei der Nachricht beispielsweise um eine Remotebenachrichtigung handelt, wird sie dem Benutzer im Infobereich angezeigt.

5. Benutzerinstallation und Interaktion mit FCM

5.1 Installation der Anwendung

5.1.1 Download und Installation

- **Google Play:** Der Benutzer lädt die App aus dem Google Play Store herunter.

- **Installation:** Nach dem Download wird die App automatisch auf dem Gerät des Benutzers installiert.

5.1.2 Erste App-Initialisierung

- **Start:** Der Benutzer öffnet die App zum ersten Mal, indem er auf das App-Icon tippt.

- **Berechtigungen:** Beim ersten Start kann die App verschiedene Berechtigungen anfordern, die für ihre Funktionalität notwendig sind. Dazu gehören:

    - **Benachrichtigungen**: Berechtigung, Push-Benachrichtigungen an das Gerät zu senden. Dies ist entscheidend für die Nutzung von FCM.

    - **Standortzugriff:** Falls die App standortbasierte Dienste anbietet.

    - **Speicherzugriff:** Falls die App Dateien auf dem Gerät speichern oder lesen muss.

5.2 Benutzeranmeldung und Interaktion mit Firebase

5.2.1 Anmeldung

- Die App bietet verschiedene Methoden zur Benutzeranmeldung, die über Firebase Authentication abgewickelt werden. Dies sorgt für eine sichere und einfache Benutzerverwaltung.

    - **Anmeldung:** Der Benutzer meldet sich mit seiner verifizierten E-Mail-Adresse und dem Passwort an.

    - **Firebase Authentication:** Firebase überprüft die Anmeldedaten und stellt ein Authentifizierungstoken bereit, das die App zur Identifizierung des Benutzers verwendet.

5.2.2 Interaktion mit Firebase-Diensten

- **Cloud Firestore/Realtime Database:** Sobald der Benutzer authentifiziert ist, kann die App mit Firebase-Datenbanken interagieren, um benutzerbezogene Daten zu speichern und abzurufen.

- **Cloud Storage:** Die App kann Dateien wie Bilder und Videos in Firebase Cloud Storage hochladen und verwalten.

- **Firebase Analytics:** Die App sendet Ereignisdaten an Firebase Analytics, um das Benutzerverhalten zu verfolgen und Berichte zu erstellen.

5.3 Interaktion mit FCM

5.3.1 FCM-SDK Integration

- **SDK Einbindung:** Die Integration des FCM-SDKs in die App ist notwendig, um Push-Benachrichtigungen zu ermöglichen. Dies umfasst das Hinzufügen der entsprechenden Konfigurationsdateien und die Einbindung der notwendigen Bibliotheken in das Projekt.

5.3.2 Initialisierung des Dienstes

- **FirebaseApp Initialisierung:** Die App muss Firebase initialisieren, um FCM nutzen zu können. Dies erfolgt während des App-Starts.

- **Konfiguration:** Die Konfigurationsdateien, die von der Firebase-Konsole heruntergeladen wurden, enthalten die notwendigen Informationen, um die App mit dem Firebase-Projekt zu verbinden.

- **Authentifizierung:** Die App authentifiziert sich bei Firebase, um auf die Cloud-Messaging-Dienste zugreifen zu können.

- **Token Abrufen:** Nach der Initialisierung fordert die App ein Registrierungstoken von FCM an. Dieses Token dient als eindeutiger Identifikator für das Gerät oder die App-Instanz.

5.3.3 Registrierungstoken Handhabung

- **Token Abrufen:** Das FCM-SDK fordert automatisch ein Registrierungstoken an und stellt es der App zur Verfügung, sobald es erstellt oder aktualisiert wird.

- **Token Gültigkeit:** Das Registrierungstoken ist solange gültig, bis es explizit zurückgezogen oder aktualisiert wird.

- **Token Speichern:** Das Registrierungstoken wird lokal auf dem Gerät gespeichert und gegebenenfalls an den Server gesendet, um Push-Benachrichtigungen an das Gerät zu ermöglichen.

- **Sichere Speicherung:** Das Token sollte sicher gespeichert werden, um Missbrauch zu verhindern.

5.3.4 Senden des Tokens an den Server

- **API-Endpunkt:** Die App sendet das Registrierungstoken an einen speziellen API-Endpunkt auf dem Server. Dieser Endpunkt ist dafür zuständig, die Tokens zu empfangen und zu speichern.

- **Sichere Übertragung:** Die Übertragung des Tokens sollte über eine gesicherte Verbindung (HTTPS) erfolgen.

- **Zusätzliche Daten:** Zusammen mit dem Token können auch andere relevante Informationen wie Benutzer-ID, App-Version oder Gerätetyp an den Server gesendet werden.

- **Datenvalidierung:** Der Server validiert die empfangenen Daten, um sicherzustellen, dass sie korrekt und vollständig sind.

-

5.4 FCM-Nachrichten

Firebase Cloud Messaging (FCM) bietet eine breite Palette von Messaging-Optionen und -Funktionen.

Um das Firebase Admin SDK zu implementieren, benötigen Sie die folgenden Anmeldedaten aus Ihrem Firebase Projekt:

- **Projekt-ID**
  Eine eindeutige Kennung für Ihr Firebase-Projekt, die in Anfragen an den HTTP-Endpunkt FCM v1 verwendet wird. Dieser Wert ist in der Firebase Console im Bereich Einstellungen verfügbar.

- **Registrierungstoken**
  Ein eindeutiger Tokenstring, der die einzelnen Client-App-Instanzen identifiziert. Das Registrierungstoken ist für Nachrichten an einzelne Geräte und in Gerätegruppen erforderlich. Registrierungstokens müssen geheim gehalten werden.

- **Absender-ID**
  Ein eindeutiger numerischer Wert, der beim Erstellen Ihres Firebase-Projekts erstellt wird und in der Firebase Console im Bereich Einstellungen auf dem Tab Cloud Messaging verfügbar ist. Die Sender-ID wird verwendet, um jeden Absender zu identifizieren, der Nachrichten an die Clientanwendung senden kann.

- **Zugriffstoken**
  Ein kurzlebiges OAuth 2.0-Token, das Anfragen an die HTTP v1 API autorisiert. Dieses Token ist mit einem Dienstkonto verknüpft, dass zu Ihrem Firebase-Projekt gehört. Zum Erstellen und Rotieren von Zugriffstokens führen Sie die unter Sendeanfragen autorisieren beschriebenen Schritte aus.

5.5 FCM-Registrierungstoken-Verwaltung

Bei der Verwendung von FCM-APIs können Ressourcen verschwendet werden, wenn Nachrichten an inaktive Geräte mit veralteten Registrierungstokens gesendet werden. Dies kann zu fehlerhaften Zustellungsdaten und scheinbar niedrigen Zustellungsraten führen. In diesem Leitfaden werden Maßnahmen zur effizienten Nachrichtenausrichtung und zur Sicherstellung gültiger Zustellungsberichte erläutert.

5.5.1 Veraltete und abgelaufene Registrierungstoken

Veraltete Tokens sind mit Geräten verknüpft, die seit über einem Monat keine Verbindung zum FCM hatten. Mit zunehmender Inaktivität sinkt die Wahrscheinlichkeit, dass diese Geräte jemals wieder eine Verbindung herstellen. Tokens, die 270 Tage inaktiv waren, gelten als abgelaufen und werden von FCM als ungültig markiert. Falls das Gerät erneut eine Verbindung herstellt, wird ein neues Token erstellt.

Veraltete Tokens entstehen, wenn Geräte verloren gehen, zerstört oder vergessen werden. Abgelaufene Tokens werden von FCM abgelehnt, aber bei erneuter Verbindung wird ein neues Token ausgestellt.

5.5.2 Grundlegende Best Practices

Bei der Verwendung von FCM-APIs sollten Sie folgende Best Practices beachten:

1) Registrierungstokens verwalten: Rufen Sie Tokens von FCM ab, speichern Sie sie auf Ihrem Server und halten Sie eine aktuelle Liste der aktiven Tokens. Implementieren Sie einen Token-Zeitstempel und aktualisieren Sie diesen regelmäßig.

2) Veraltete Tokens entfernen: Entfernen Sie Tokens, die FCM als ungültig markiert, und überwachen Sie Anzeichen für veraltete Tokens, um diese proaktiv zu entfernen.

5.5.3 Registrierungstoken abrufen und speichern

Beim ersten Start Ihrer App generiert das FCM SDK ein Registrierungstoken für die Client-App-Instanz. Dieses Token sollten Sie beim ersten Start abrufen und zusammen mit einem Zeitstempel auf Ihrem App-Server speichern. Der Zeitstempel muss von Ihrem Code und Ihren Servern implementiert werden, da er von FCM SDKs nicht bereitgestellt wird.

Speichern und aktualisieren Sie das Token auf dem Server, wenn:

- Die App auf einem neuen Gerät wiederhergestellt wird
- Der Benutzer die App deinstalliert und erneut installiert
- Der Benutzer App-Daten löscht
- Die App nach Ablauf des Tokens wieder aktiv wird

5.5.4 Aktualität der Tokens sicherstellen und veraltete Tokens entfernen

Um festzustellen, ob ein Token veraltet ist, setzen Sie einen Schwellenwert fest. FCM betrachtet Tokens als veraltet, wenn ihre App-Instanz einen Monat lang keine Verbindung hergestellt hat. Ein aktives Gerät hätte sein Token aktualisiert. Abhängig von Ihrem Anwendungsfall kann dieser Zeitraum variieren.

5.5.5 Tokens regelmäßig aktualisieren

Aktualisieren Sie regelmäßig alle Registrierungstokens auf Ihrem Server. Dies erfordert:

1. Client-App-Logik: Fügen Sie Ihrer App Logik hinzu, um das aktuelle Token abzurufen (z. B. getToken() für Android) und senden Sie es zur Speicherung an Ihren Server, inklusive Zeitstempel. Dies könnte monatlich geschehen.

2. Server-Logik: Aktualisieren Sie den Zeitstempel der Tokens regelmäßig, auch wenn sich das Token nicht geändert hat.

6. Serverumgebung und FCM

Die Serverseite von Firebase Cloud Messaging besteht aus zwei Komponenten:

1) **Das von Google bereitgestellte FCM-Backend:**
   - Zuständig für das Empfangen, Verarbeiten und Weiterleiten von Nachrichten.

2) **Der App-Server oder eine andere vertrauenswürdige Serverumgebung:**
   - Führt die Serverlogik aus und kommuniziert mit dem FCM-Backend.

6.1 Anforderungen an die vertrauenswürdige Serverumgebung

FCM-Kommunikation: Die Serverumgebung muss Nachrichtenanfragen an das FCM-Backend senden können.
Exponentieller Backoff: Die Umgebung sollte Anfragen mit exponentiellem Backoff erneut senden können.
Sicherer Speicher: Sie muss Server-Anmeldeinformationen und Client-Tokens sicher speichern können.

API-Endpunkt: Ein API-Endpunkt muss HTTP-POST-Anfragen empfangen und JSON-Daten verarbeiten können.
JSON-Verarbeitung: Der Endpunkt muss JSON-Daten im Anfrage-Body verarbeiten können
Datenbankintegration: Eine Datenbank ist nötig, um Tokens und Zusatzinfos wie Benutzer-ID zu speichern.

6.2 Firebase Admin SDK für FCM

Das Firebase Admin SDK wird aufgrund seiner Unterstützung gängiger Programmiersprachen und praktischer Authentifizierungs- und Autorisierungsmethoden empfohlen. Das Admin SDK unterstützt **Node.js, Java, Python, C#** und **Go**.

Die Admin-FCM-API übernimmt die Authentifizierung beim Backend und erleichtert das Senden von Nachrichten und die Verwaltung von Themenabonnements. Mit dem Firebase Admin SDK können Sie:

- Senden Sie Nachrichten an einzelne Geräte
- Senden Sie Nachrichten an Themen und Bedingungsanweisungen, die einem oder mehreren Themen entsprechen.
- Geräte für Themen abonnieren und abmelden
- Erstellen Sie Nachrichtennutzlasten, die auf verschiedene Zielplattformen zugeschnitten sind

6.3 Voraussetzungen

Stellen Sie sicher, dass Sie über eine Server-App verfügen.
Das Firebase Admin SDK muss in der Server-App installiert und konfiguriert sein.
Stellen Sie sicher, dass auf Ihrem Server Folgendes ausgeführt wird: **Admin .NET SDK – .NET Framework 4.6.1+ oder .NET Standard 2.0 für .Net Core 2.0+**

7. Integration des Firebase Admin SDK in eine .NET-Konsolen App

7.1 Einführung

Eine Konsolen-App im .NET-Framework wird als Server verwendet, um mit Firebase zu kommunizieren und Tokens zu verwalten.

7.2 Schritt-für-Schritt-Anleitung

1) Zur Projektmappe ein neues Projekt als Konsolen-App hinzufügen

2) ***NuGet-Pakete*** installieren:

3) Den entsprechenden Code in die Klasse ***Program.cs*** kopieren

```
using Google.Apis.Auth.OAuth2;
using FirebaseAdmin;
using FirebaseAdmin.Messaging;

class Program
{
  static async Task Main(string[] args)
  {
    try
    {
      string projectId = " ";          // Projekt-ID
      string credentialsPath = " ";            //Pfad zur JSON-Datei

      Environment.SetEnvironmentVariable("GOOGLE_APPLICATION_CREDENTIALS", credentialsPath);

      FirebaseApp.Create(new AppOptions()
      {
        Credential = GoogleCredential.GetApplicationDefault(),
        ProjectId = projectId
      });
```

```
            Console.WriteLine("Firebase Admin SDK wurde initialisiert.");


            string registrationToken = " ";            //Zugriffstoken

            if (string.IsNullOrEmpty(registrationToken))
            {
               Console.WriteLine("Ungültiges Token. Die Nachricht konnte nicht gesendet werden.");
               return;
            }

            var message = new Message
            {
               Notification = new Notification
               {
                  Title = "Test1",
                  Body = "Nachricht vom Server!"
               },
               Token = registrationToken
            };

            string response = await
FirebaseMessaging.DefaultInstance.SendAsync(message).ConfigureAwait(false);
            Console.WriteLine("Nachricht erfolgreich gesendet. Nachrichten-ID: " + response);
         }
         catch (FirebaseMessagingException ex)
         {
            Console.WriteLine($"Fehler beim Senden der Nachricht: {ex.ErrorCode} - {ex.Message}");
         }
      }
   }
```

➢ Nachrichten senden über den Server

Um Nachrichten über den Server zu senden, müssen die folgenden Daten zu den kommentierten Zeilen des obigen Codes hinzugefügt werden:

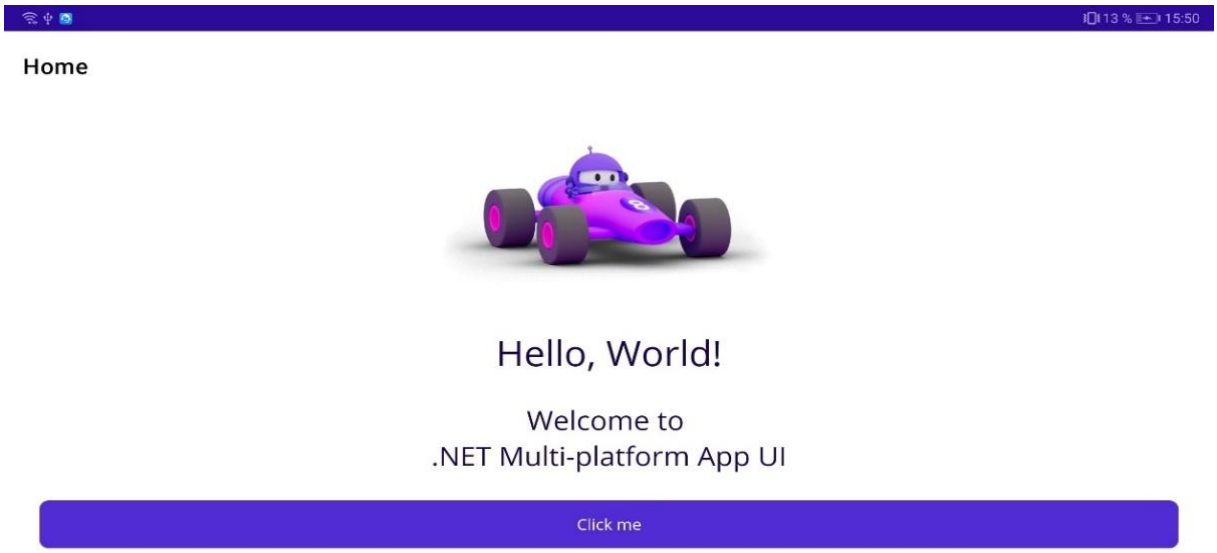
- Projekt-ID
  Projektübersicht > Projekteinstellungen > Allgemein

- JSON-Datei
  Projektübersicht > Projekteinstellungen > Dienstkonten > Neuen privaten Schüssel generieren
  Die JSON-Datei muss sicher gespeichert werden, der Pfad kopiert und in den Code eingefügt.

- Zugriffstoken

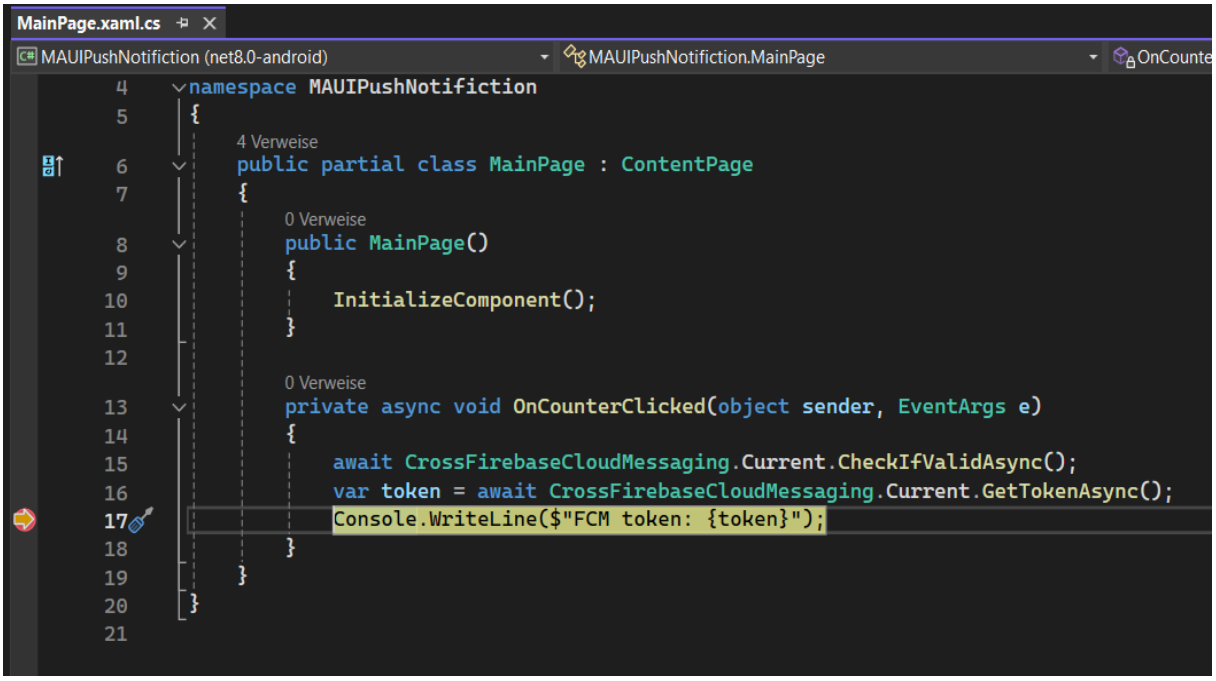
In der Klasse ***MainPage.xaml.cs*** wird ein Haltepunkt in der Zeile ***Console.WriteLine($"FCM token: {token}")*** gesetzt.

Nach dem Start des Programms wird die .NET MAUI-Anwendung auf dem externen Gerät/Emulator installiert und anschließend das Token generiert. Um ein neues Token zu generieren, muss die Anwendung vom Gerät deinstalliert werden, andernfalls generiert die Anwendung immer dasselbe Token.



Das Token erscheint auf der Zeile, in der der Haltepunkt gesetzt wurde, mit gelbem Hintergrund. Von dort aus kann es kopiert und verwendet werden.

Um mit dem Versenden einer Nachricht beginnen zu können, muss zunächst der Zugriffstoken generiert werden. Danach können Nachrichten vom Server nur dann gesendet werden, wenn alle Daten im Server korrekt eingegeben wurden.



8. Literaturverzeichnis

Im Rahmen der Projektrealisierung wurden folgende bibliografische Quellen und Online-Ressourcen herangezogen:

- IBM: Push-Benachrichtigungen
- Microsoft: .NET MAUI
- GitHub-Repository
- Introducing Firebase
- Firebase Cloud Messaging
- FCM -Architektur
- Introducing Firebase Cloud Messaging
- Firebase Cloud Messaging

- Microsoft Learn
- FCM Nachrichten
- FCM-Registrierungstoken-Verwaltung
- Serverumgebung und FCM
- Best Practices für die Verwaltung von Dienstkontoschlüsseln
- Das Firebase Admin SDK zum Server hinzufügen

Diese Quellen wurden konsultiert und genutzt, um die notwendigen Informationen für Design, Implementierung und Dokumentation zu erhalten und so einen soliden Rahmen für die Projektentwicklung bereitzustellen.