#21 TECHARTIKEL

Push-Benachrichtigungen mittels Firebase in .NET-Webanwendungen

22.09.2025

Jakob Bonhoeffer

Inhalt

Push-Benachrichtigungen mittels Firebase in .NET-Webanwendungen

Firebase ist eine umfassende Entwicklungsplattform von Google, die speziell für Web- und Mobile-Anwendungen konzipiert wurde. Sie stellt eine Reihe cloudbasierter Dienste bereit, die viele typische Backend-Aufgaben übernehmen. Dazu gehören unter anderem Echtzeitdatenbanken, Benutzer-Authentifizierung, Hosting für Webanwendungen, serverlose Funktionen über Cloud Functions sowie Tools zur Leistungsanalyse und Fehlerberichterstattung. Besonders hervorzuheben ist Firebase Cloud Messaging (FCM), ein System zum Versenden von Push-Benachrichtigungen.

Durch Firebase entfällt der Aufwand für den Aufbau und Betrieb einer komplexen Serverinfrastruktur. Die Plattform ist dadurch nicht nur bei Start-ups und in der Industrie beliebt, sondern eignet sich auch hervorragend für studentische Projekte, Prototypen und erste professionelle Anwendungen.

1. Typische Einsatzszenarien von Firebase

Firebase kann in zahlreichen Anwendungsbereichen eingesetzt werden – insbesondere dort, wo Echtzeitverarbeitung und flexible Skalierbarkeit gefragt sind. Typische Szenarien sind:

- **Nutzer-Authentifizierung:** Login via E-Mail, Google, Facebook oder anonym.
- **Daten in Echtzeit:** Realtime Database oder Cloud Firestore für Live-Chats, Dashboards oder Statusanzeigen.
- **Push-Benachrichtigungen:** Aktive Nutzeransprache per FCM.
- **Serverloser Code:** Cloud Functions zur Verarbeitung von Events, ohne eigene Server zu betreiben.
- **Hosting und Deployment:** Einfaches Hosten von SPAs oder statischen Seiten.
- **Monitoring & Analytics:** Absturzberichte, Nutzerverhalten und Performanceanalyse.

Gerade bei mobilen oder interaktiven Webanwendungen spielen diese Funktionen ihre Stärken aus.

2. Integration in eine .NET-Webanwendung

Die Integration von Firebase in eine serverseitige .NET-Webanwendung erfolgt in der Regel über das Firebase Admin SDK. Damit lässt sich auf verschiedene Firebase-Dienste zugreifen. Die Authentifizierung erfolgt dabei über ein Servicekonto.

2.1. Voraussetzungen

Vor Beginn sind einige Vorbereitungen notwendig:

- Firebase-Projekt in der Firebase Console anlegen
- Servicekonto erstellen und die JSON-Datei mit dem privaten Schlüssel herunterladen
- NuGet-Paket `FirebaseAdmin` installieren:

```
Install-Package FirebaseAdmin
```

2.2. Initialisierung des Admin SDKs

Im nächsten Schritt wird das SDK initialisiert, um die Firebase-Dienste nutzen zu können:

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

FirebaseApp.Create(new AppOptions()
{
    Credential = GoogleCredential.FromFile("path/to/serviceAccountKey.json"),
});
```

Diese Initialisierung erfolgt idealerweise beim Start der Anwendung, etwa in `Program.cs`.

Aufbau der serviceAccountKey.json-Datei

Die JSON-Datei enthält die Zugangsdaten zum Servicekonto. Eine typische Datei sieht folgendermaßen aus:

```
{
  "type": "service_account",
  "project_id": "your-project-id",
  "private_key_id": "somekeyid",
  "private_key": "-----BEGIN PRIVATE KEY-----\n...\n-----END PRIVATE KEY-----\n",
  "client_email": "firebase-adminsdk@your-project-id.iam.gserviceaccount.com",
  "client_id": "...",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "..."
}
```

Diese Datei erhält man in der Firebase Console unter **Projektübersicht > Einstellungen > Servicekonten > Neuen privaten Schlüssel generieren**. Sie sollte sicher gespeichert und niemals in Versionskontrollsysteme eingecheckt werden.

2.3. Beispiel: Senden einer einfachen Push-Nachricht

Nachdem das SDK eingerichtet ist, lässt sich eine Nachricht an ein spezifisches Gerät senden (vorausgesetzt, es besitzt einen gültigen FCM-Token):

```
using FirebaseAdmin.Messaging;

var message = new Message()
{
    Notification = new Notification
    {
        Title = "Hallo!",
        Body = "Dies ist eine Testnachricht"
    },
    Token = "device_fcm_token"
};

string response = await FirebaseMessaging.DefaultInstance.SendAsync(message);
Console.WriteLine("Erfolgreich gesendet: " + response);
```

Dieser Code eignet sich für erste Tests und Debugging.

2.4. Nachrichten an bestimmte Benutzergruppen versenden

Firebase erlaubt es, Nachrichten gezielt an Gruppen von Geräten zu senden. Dafür gibt es mehrere Mechanismen:

Topics (Themen)

Nutzer können sich über ihr Gerät bestimmten Themen zuordnen:

```
var message = new Message()
{
    Notification = new Notification
    {
        Title = "Sport News",
        Body = "Neues Spiel heute Abend!"
    },
    Topic = "sports"
};

await FirebaseMessaging.DefaultInstance.SendAsync(message);
```

Gerätegruppen

Mehrere Geräte eines Nutzers können zu einer gemeinsamen Gruppe zusammengefasst werden. Dies ist insbesondere für Multidevice-Nutzer sinnvoll.

Konditionale Ausdrücke

Es ist auch möglich, mit logischen Bedingungen gezielt mehrere Topics zu kombinieren:

```
var message = new Message()
{
    Notification = new Notification
    {
        Title = "Exklusiv",
        Body = "Nur für Premium-Nutzer"
    },
    Condition = "'sports' in topics && 'premium' in topics"
};

await FirebaseMessaging.DefaultInstance.SendAsync(message);
```

Weitere Beispiele für konditionale Ausdrücke:

- Nur Nutzer, die sich entweder für “news” oder “updates” interessieren:
  `Condition = "'news' in topics || 'updates' in topics"`
- Nutzer, die sowohl “music” als auch “concerts” abonniert haben:
  `Condition = "'music' in topics && 'concerts' in topics"`
- Nutzer, die “beta” abonniert, aber nicht “trial” abonniert haben:
  `Condition = "'beta' in topics && !('trial' in topics)"`
- Nutzer, die weder “ads” noch “tracking” aktiviert haben:
  `Condition = "!('ads' in topics) && !('tracking' in topics)"`

Diese Bedingungen ermöglichen eine sehr zielgerichtete Ansprache und reduzieren unnötige Benachrichtigungen.

2.5. Nutzer einem Topic zuweisen (Client und Server)

Damit ein Gerät Nachrichten eines bestimmten Topics empfangen kann, muss es diesem Topic zugewiesen werden. Dies kann sowohl client- als auch serverseitig geschehen:

Clientseitig (Android)

Auf Android geschieht die Anmeldung zum Topic mithilfe des Firebase SDK:

```
FirebaseMessaging.getInstance().subscribeToTopic("sports")
    .addOnCompleteListener(task -> {
        if (task.isSuccessful()) {
            Log.d("Firebase", "Erfolgreich beim Topic angemeldet.");
        } else {
            Log.e("Firebase", "Fehler bei der Anmeldung zum Topic.");
        }
    });
```

Dieser Code wird typischerweise nach der Authentifizierung oder bei einer bestimmten Nutzeraktion ausgeführt.

Serverseitig (C#)

Auch serverseitig können Geräte Topics zugewiesen werden:

```
var tokens = new List<string>
{
    "device_fcm_token_1",
    "device_fcm_token_2"
};

await FirebaseMessaging.DefaultInstance.SubscribeToTopicAsync(tokens, "sports");
```

Genauso einfach können Geräte mit `UnsubscribeFromTopicAsync` wieder entfernt werden. So lässt sich die Kommunikation gezielt steuern.

2.6. Verwaltung von FCM-Tokens

Um Push-Benachrichtigungen effektiv nutzen zu können, ist es wichtig, die FCM-Tokens der Geräte zentral zu verwalten. Dies kann serverseitig beispielsweise wie folgt realisiert werden:

- **Speicherung in einer Datenbank**: Beim ersten Start oder Login sendet der Client seinen aktuellen Token an den Server. Dieser speichert den Token gemeinsam mit der Benutzer-ID oder Geräte-ID in einer relationalen Datenbank oder einem Dokumentenspeicher wie Firestore oder MongoDB.
- **Regelmäßiges Aktualisieren**: Da sich Tokens ändern können (z. B. bei Neuinstallation oder Token-Rotation), sollte der Client seinen Token bei jeder Änderung erneut übermitteln. Das Firebase SDK bietet dazu Events wie `onNewToken()` (Android).
- **Zuordnung zu Topics und Nutzergruppen**: Die gespeicherten Tokens können genutzt werden, um gezielt Nachrichten an einzelne Nutzer,

Gruppen oder Topics zu senden. Auch eine gezielte Abmeldung oder Löschung ungültiger Tokens wird dadurch möglich.

Ein einfaches Beispiel für das Speichern eines Tokens in einer .NET-WebAPI:

```
[HttpPost("/register-token")]
public IActionResult RegisterToken([FromBody] TokenRegistrationDto dto)
{
    // dto enthält: UserId, DeviceId, Token
    _tokenService.StoreOrUpdate(dto.UserId, dto.DeviceId, dto.Token);
    return Ok();
}
```

Beispielhafte Tabelle zur Tokenverwaltung

UserId	DeviceId	FCM Token	Letzte Aktualisierung
12345	abc123	fcm_token_1	2025-06-25 10:42
12345	def456	fcm_token_2	2025-06-24 18:10
67890	ghi789	fcm_token_3	3:32

Diese Tabelle könnte beispielsweise in einer SQL-Datenbank oder einem Firestore-Dokumentenspeicher gepflegt werden.

2.7. Weitere Nutzungsmöglichkeiten in .NET-Projekten

Firebase bietet noch viele weitere Anwendungsmöglichkeiten für .NET-Projekte:

- Benutzerverwaltung über Firebase Authentication
- Zugriff auf Firestore-Datenbanken per REST oder Drittanbieterbibliotheken
- Deployment von Frontend-Anteilen mit Firebase Hosting (z. B. Angular, React)

3. Fazit

Firebase ist ein leistungsfähiges Werkzeug zur Entwicklung moderner Web- und Mobile-Anwendungen. Es bietet skalierbare Backend-Funktionen und reduziert den Entwicklungsaufwand erheblich. Für .NET-Projekte lassen sich Push-Benachrichtigungen effizient und zielgerichtet umsetzen, auch differenziert nach Nutzergruppen. Dank klarer Dokumentation und vielseitiger Möglichkeiten eignet sich Firebase ideal für den produktiven Einsatz ebenso wie für den Projektstart und das Lernen moderner Entwicklungstechniken.