Visual Studio App Center - Demos

Ausgangslage

- **`Xamarin.Forms`**-App («MUG App Center»)
  - UI: 1 Label + 4 Buttons, davon einer mit bestehender Funktion
  - Keine Integration der VS App Center SDKs
- **`Xamarin.UITest`**-Test
  - Startup & Verhalten von «Button 1» (Popup)
- Ein Script zur Ausführung der UI Tests in App Center (**`Build/ui_tests.sh`**)
- Keystore-Datei für Signierung der Android-App (**`Build/Android.keystore`**)

Keystore Password	**`11111111`**
Key Alias	**`MUGAppCenter`**
Key Password	**`11111111`**

Projekt erstellen

- Dashboard > Add new app

App Name	Hello App Center
Description	(leer)
Icon	(optional)
OS	Android
Platform	Xamarin

Build einrichten

- Optional: Fork der Ausgangslage erzeugen (für Zugriff mit Noser-Account)
- Build > Select a service > GitHub > **`MUG_Zentralschweiz`**
- Branches > **`talk/vs-app-center`** > Configure build

Project	**`MUGAppCenter.UI.Android.csproj`**
Configuration	Release
SDK Version	**`Xamarin.Android 9.0`**
Build Scripts	None
Build Frequency	Build this branch on every push
Automatically incr.	On / Build ID
Environment Vars	Off
Sign Builds	On
Test on Real Device	Off
Distribute Builds	On / Collaborators
Build Status Badge	On

Distribution zeigen

- Aufgrund der Build-Option *«Distribute Builds»* werden Builds bereits verteilt
  - Login auf VS App Center via Smartphone
  - Installation der vorhin erzeugten App möglich
  - Email kommt nach einigen Minuten bis Stunden beim Tester an
- Distribute > Groups > Add Group

Group Name	Beta Tester
Allow Public Access	Off
Testers	Thomas Kälin Loana Albisser

- Distribute > Groups > Distribute Release > Choose Build from branch > **`talk/vs-app-center`**
- Distribute > Stores > Connect to Store
  - Übermittlung direkt an den Google Play Store möglich
  - Das funktioniert auch innerhalb eines «Builds» → Tipp: Release-Branch konfigurieren

Testing einrichten

- Nutzen der Build-Option *«Test on Real Device»* erklären
  - Launch Test auf einem Gerät inkl. Screenshot
  - Funktioniert auch ohne bezahlte «Test»-Option
  - Aber: dies erhöht die Build-Dauer um ca. 5-10 Minuten! → Guthaben beachten!
- Funktionalität der *«Test»*-Funktionalität erklären
  - Konfigurator einen CLI-Aufruf, der manuell ausgeführt werden muss (!)
  - Testprojekte können aktuell NICHT im VS App Center gebaut und ausgeführt werden
  - Eine Integration wäre evtl. via Build Scripts möglich
- Den Wizard zeigen inkl. generiertem CLI-Befehl zeigen (Test > Start Testing your app)

Devices	(1) Google Pixel 3 (Android 9.0) (2) Samsung Galaxy S9 (Android 8.0) (3) Huawei P10 (Android 7.0)
Test Series	**`master`**
System Language	German (Germany)
Test Framework	**`Xamarin.UITest`**

- Das Script **`Build/ui_tests.sh`** erklären
  - Schritt 1: Android-APK erstellen
  - Schritt 2: Test-Projekt erstellen
  - Schritt 3: via CLI einloggen und anschliessend Ausführung starten
- In App Center ein API Token erstellen und im Script einfügen, anschliessend Script starten.
  - Erstellung des Tokens: *Account* > *API-Tokens*
  - Platzhalter im Script: **`APPCENTER_TOKEN=…`**
- In App Center kann unter *«Test runs»* der Fortschritt verfolgt werden
  - Ausführung dauert ca. 5-10 Minuten (je nach Verfügbarkeit der Geräte)

SDK integrieren

- NuGet-Packages zu allen Projekten hinzufügen (Demo mit Version **`1.13.0`**)
  - **`Microsoft.AppCenter`**
  - **`Microsoft.AppCenter.Analytics`**
  - **`Microsoft.AppCenter.Crashes`**
  - **`Microsoft.AppCenter.Push`**
- Benötigte **`using`**-Statements in der **`App`**-Klasse ergänzen

```
using Microsoft.AppCenter;
using Microsoft.AppCenter.Analytics;
using Microsoft.AppCenter.Crashes;
using Microsoft.AppCenter.Push;
```

- Die Methode **`App.OnStart()`** anpassen. Das **`{Secret}`** findet sich auf der Overview-Seite.

```
AppCenter.Start("android={Secret};" +
                "uwp={Secret};" +
                "ios={Secret}",
                typeof(Analytics),
                typeof(Crashes),
                typeof(Push));
```

- Einen Git-Commit mit den Änderungen erzeugen, noch keinen Push ausführen

Analytics einrichten

- Durch Integration der SDK werden bereits automatisch verschiedene Daten gesammelt
  - Beispiele: Geräte-Typ, OS-Version, Session Tracking, etc.
  - Achtung: Android- & iOS-Apps sind voneinander getrennt. Eine übergreifende Auswertung via Export zu *«Azure Application Insights»* möglich.
- Zusätzlich können manuell *«Events»* eingebaut werden
  - Max. 200 unterschiedliche Events
  - Max. 256 Zeichen für den Namen des Events
  - Max. 125 Zeichen für Eigenschaften und deren Werte
- Anpassung der Methode **`MainPage.DoOnTrackEventPressed()`**

```
using Microsoft.AppCenter.Analytics;

Analytics.TrackEvent("Der 'Track'-Button wurde gedrückt.");
```

- App starten und den Button mehrfach drücken
- Im App Center die *«Analytics»*-Page öffnen
  - *«Overview»* enthält Daten zu Nutzern / Ländern / Geräten
  - *«Events»* zeigt die Auswertung der Events an
  - *«Log Flow»* sollte die Events in «Echtzeit» darstellen, hat aber nie funktioniert
- Einen Git-Commit mit den Änderungen erzeugen und einen Push ausführen.

Diagnostics einrichten

- Die SDKs wurden in einem früheren Schritt bereits integriert und aktiviert.
    - Achtung: Android- & iOS-Apps sind voneinander getrennt. Eine übergreifende Auswertung via Export zu *«Azure Application Insights»* möglich.
- Anpassung der Methode **`MainPage.DoOnTrackExceptionPressed()`**

```
using Microsoft.AppCenter.Crashes;

var exception = new Exception("Der 'Exception'-Button wurde gedrückt.");
Crashes.TrackError(exception);
```

- App starten und den Button mehrfach drücken
- Im App Center die *«Diagnostic»*-Page öffnen
    - Nach einigen Minuten erscheint die Exception als *«Error»* in der Auflistung
    - Zu der Exception wird automatisch ein *«Issue»* eröffnet.
    - Verknüpfung mit externen Bug-Trackern möglich (z.B. GitHub): *Settings > Services*
- Anpassung der Methode **`MainPage.DoOnSimulateCrashPressed()`**

```
throw new Exception("Der 'Crash'-Button wurde gedrückt.");
```

- App starten und den Button einmalig drücken. Die App stürzt ab. Sicherheitshalber die App neu starten, damit der Crash-Report garantiert zum App Center übermittelt wird.
- Im App Center die «Diagnostic»-Page öffnen
    - Nach einigen Minuten erscheint die Exception als *«Crash»* in der Auflistung
    - Zu der Exception wird automatisch ein *«Issue»* eröffnet.
- Überschreiben der Methode **`MainPage.OnAppearing()`**

```
protected async override void OnAppearing()
{
    base.OnAppearing();

    if (await Crashes.HasCrashedInLastSessionAsync())
    {
        await DisplayAlert(string.Empty, "Sorry für den Crash!", "Ok");
    }
}
```

- App starten und den Button einmalig drücken. Die App stürzt ab. Die App erneut starten. Es erscheint ein Popup-Dialog mit einer Entschuldigung für den Crash.
- Einen Git-Commit mit den Änderungen erzeugen und einen Push ausführen.

Push Notifications einrichten

- Die SDKs wurden in einem früheren Schritt bereits integriert und aktiviert.
    - Hinweis: Push Notifications müssen für Android und iOS im Backend individuell konfiguriert werden.
- Via Firebase-Konsole ein Projekt erzeugen

Projektname	MUG App Center
Projekt-ID	**`mug-app-center`**
Speicherort	Schweiz / europe-west3

- Via Firebase-Konsole ein neues Android-App hinzufügen, anschliessend die generierte JSON-Konfiguration herunterladen und im Android-Projektverzeichnis speichern

Paketname	**`ch.mugz.mug_app_center`**
App-Alias	MUG App Center

- Die JSON-Datei zum Projekt hinzufügen, als Build-Action **`GoogleServicesJson`** verwenden
- Die Datei **`AndroidManifest.xml`** im **`<application>`**-Node ergänzen
    - Vorlage siehe App Center im *«Wizard»* unter Schritt 2
    - Achtung: Platzhalter für die *«Application ID»* mit dem Paketnamen der App ersetzen!
- App Center im *«Wizard»* unter Schritt 3 den von Firebase erhaltenen *«Server Key»* eintragen (Einstellungen > Cloud Messaging > Serverschlüssel).
- Die Methode **`App.OnStart()`** erweitern, vor dem **`Start()`**-Aufruf

```
Push.PushNotificationReceived += (sender, eventArgs) =>
{
  MainPage.DisplayAlert($"Push: {eventArgs.Title}", eventArgs.Message, "Ok");
};
```

- App starten und beenden. Einige Minuten warten, damit die Registrierung des Gerätes bei Firebase abgeschlossen werden kann. Ansonsten kommen keine Push-Notifications beim Gerät an.
- Via App Center eine Push Notification senden (*Push > Notifications > Send Notification*). Es erscheint eine System Notification. Ein Tap auf diese öffnet die App, die Event Inhalte sind aber leer (siehe Erklärung).
- Via App Center eine weitere Push Notification senden. Im App erscheint ein Popup-Dialog mit den übermittelten Werten.
- Einen Git-Commit mit den Änderungen erzeugen und einen Push ausführen.

Optionen für iOS-App erkunden

- Weiteres Projekt erstellen (selber Name, aber für iOS)
    - Ergänzung von **`App.OnStart()`** für Aktivierung der Datensammlung
    - Ausführung der App im Simulator
- Anmerkungen zu *«Build»*
    - Abhängig von *«Configuration»* unterschiedliche Steps
    - Unter *«Sign Builds»* werden die **`p12`**- und **`mobileprovision`**-Dateien hochgeladen
- Anmerkungen zu *«Distribute»*
    - Verteilung zum Apple App Store oder zu Apple TestFlight möglich
- Anmerkung zu *«Analytics»* und *«Diagnostic»*
    - Datentrennung für Android und iOS. Keine gemeinsame Auswertung direkt in App Center möglich, Export zu «Azure Application Insights» nötig.
- Anmerkungen zu *«Push»*
    - Verknüpfung mit Apple Push Notifications (APN) anstelle von Google Firebase nötig.