Xamarin.Forms × Azure 連携 Hands On Lab

C#ではじめるモバイルアプリ開発のクラウド連携ハンズオン

-Xamarin & Azure 編-

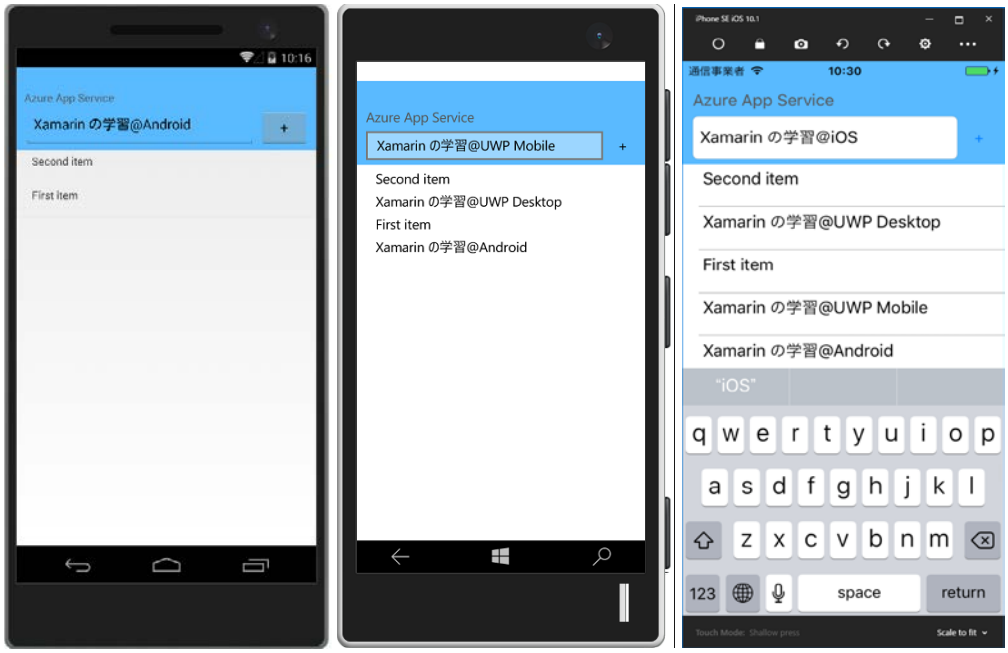
2017 年 6 月

Microsoft

内容

概要

ここでは、Azure Mobile Apps バックエンドを使用して Xamarin.Forms モバイル アプリにクラウドベースのバックエンド サービスを追加する方法を示します。新しいモバイル アプリ バックエンドと、アプリのデータを Azure に保存する簡単な Todo list Xamarin.Forms アプリの両方を作成します。完成したアプリケーションは、以下のような画面になります。



目的

このラボでは、以下の方法を示します。

- Xamarin.Forms と Azure Mobile App とを用いたクライアントアプリを実装する
- Twitter などの ID プロバイダーを使用して、Xamarin.Forms アプリのユーザーを認証する
- Xamarin.Forms アプリにプッシュ通知のサポートを追加し、Azure Notification Hubs を利用して、プッシュ通知を送信する

システム要件

このラボを完了するには、以下の環境やツールが必要です。

- アクティブな Microsoft Azure サブスクリプション。
  もしお持ちではない場合には、Visual Studio Dev Essentials プログラムをご登録ください。Azure を月間 2500 円相当分、12 か月無償でご利用いただくことができます。
  Visual Studio Dev Essential https://www.visualstudio.com/ja/dev-essentials/

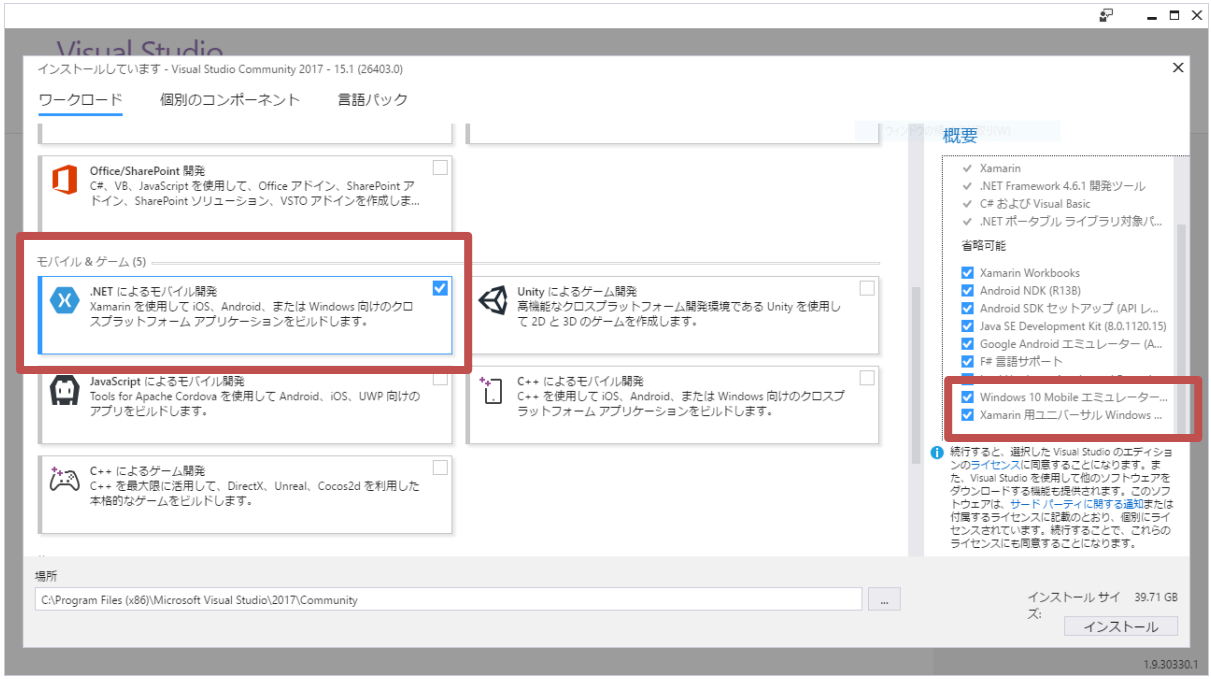
- Microsoft Windows 10 x64 Professional または Enterprise エディション
  (Hyper-V による、Windows 10 Mobile および Android エミュレータを使用するため)
- Microsoft Visual Studio 2015 Update 3 および クロスプラットフォーム開発環境
- Android デバイス（オプショナル）
- Windows 10 Mobile デバイス（オプショナル）
- Mac OS が動作する PC（オプショナル）
- iOS デバイス（オプショナル）
- Twitter アカウント
- Azure Active Directory アカウント、Facebook アカウント、Microsoft アカウント、または Google アカウント (オプショナル)

セットアップ

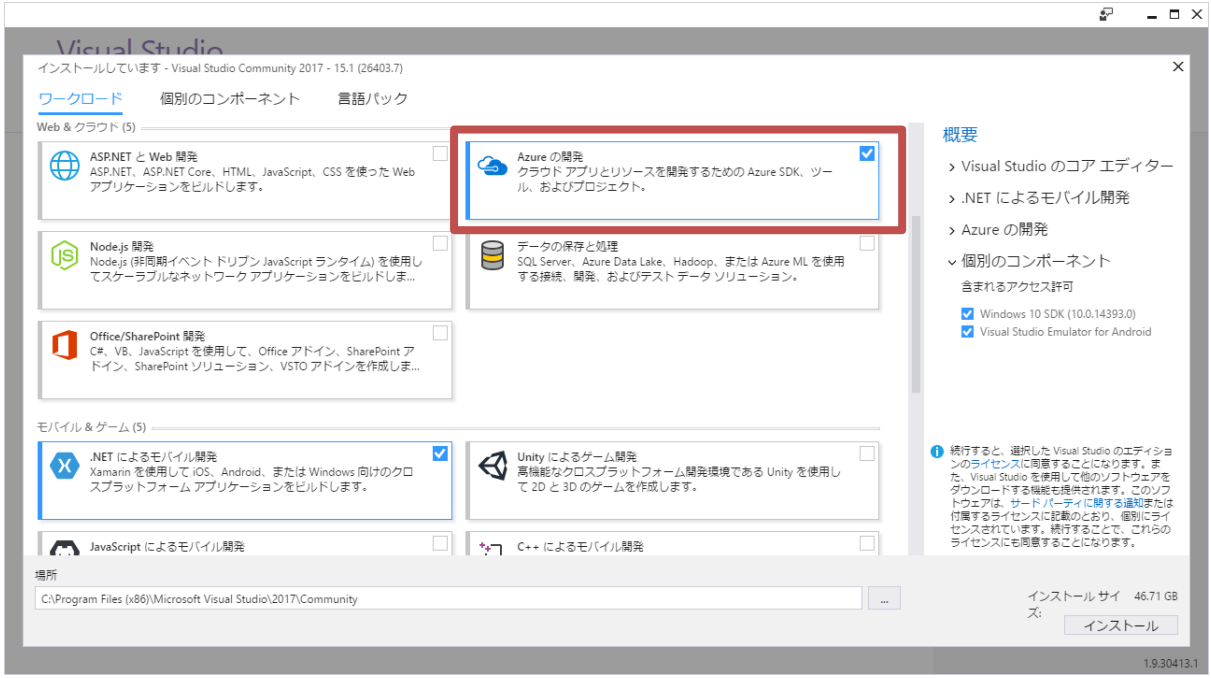
このラボ用にコンピューターの準備を整えるには、以下の手順を実行する必要があります。

1. Microsoft Windows 10 x64 Professional または Enterprise エディションをインストールします。
2. Microsoft Visual Studio 2017 のインストーラーを https://www.visualstudio.com/downloads/ よりダウンロードし、実行します。
3. Visual Studio 2017 インストーラーが起動したら、[ワークロード] 一覧より、[モバイル & ゲーム] > [.NET によるモバイル開発] を選択します。
   選択すると、右ペインの [概要] にインストールされるコンポーネントが表示されます。[省略可能] コンポーネント一覧より、下記 2 項目を選択し、チェックを入れます。

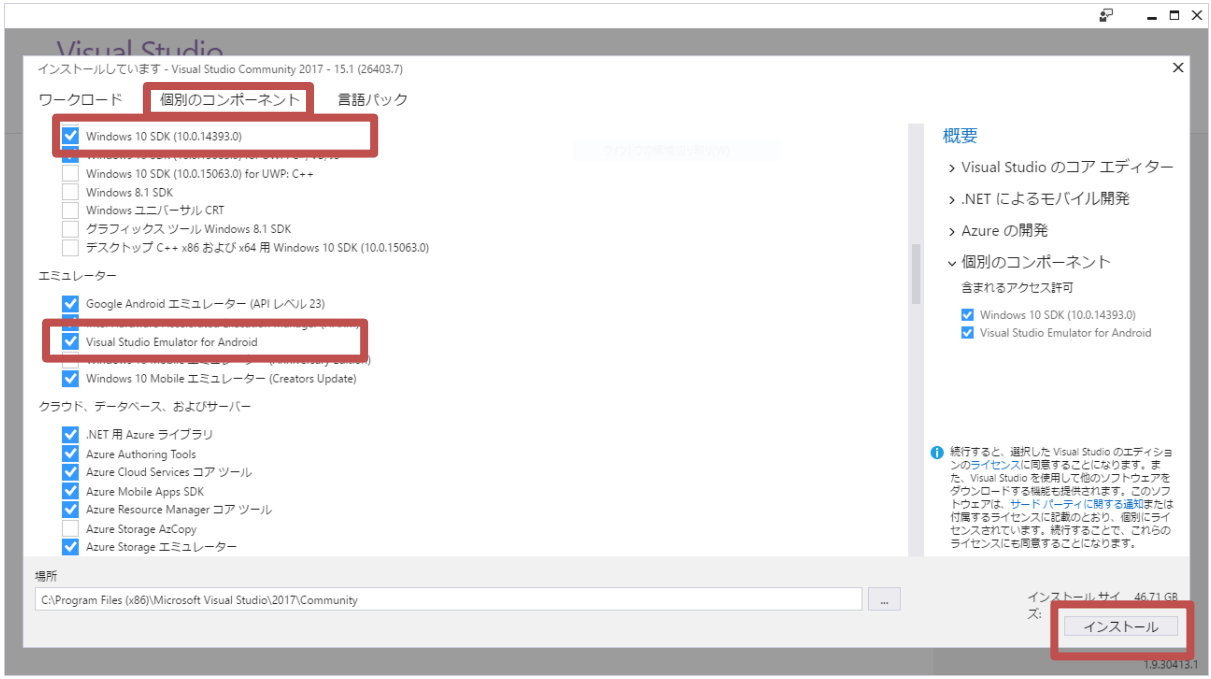
   ・Windows 10 Mobile エミュレーター (Creators Update)
   ・Xamarin 用ユニバーサル Windows プラットフォーム ツール



4. 引き続き、[ワークロード] 一覧より、[モバイル & ゲーム] > [Azure の開発] を選択します。



5. 次に、[個別のコンポーネント] タブを選択します。[SDK、ライブラリ、およびフレームワーク] > [Windows 10 SDK (10.0.14393.0)] および、[エミュレーター] > [Visual Studio Emulator for Android] を選択し、[インストール] ボタンをクリックします。



6. 各コンポーネントのダウンロードおよびインストールが完了するまで、お待ちください。インストール終了後、再起動が求められた場合は、指示に従い、PC の再起動を行なってください。

演習

このハンズオン ラボは、以下の演習から構成されています。

1. ― Xamarin.Forms アプリと Azure Mobile App バックエンドを作成する
2. ― Xamarin.Forms アプリへ認証を追加する
3. ― Xamarin.Forms アプリへのプッシュ通知を追加する

なお、演習 2 および演習 3 は、どちらも演習 1 からの継続となります。演習 2 を終了後のソースを対象に、演習 3 を行うことはできません。

ラボの推定所要時間: **90 〜 120 分**

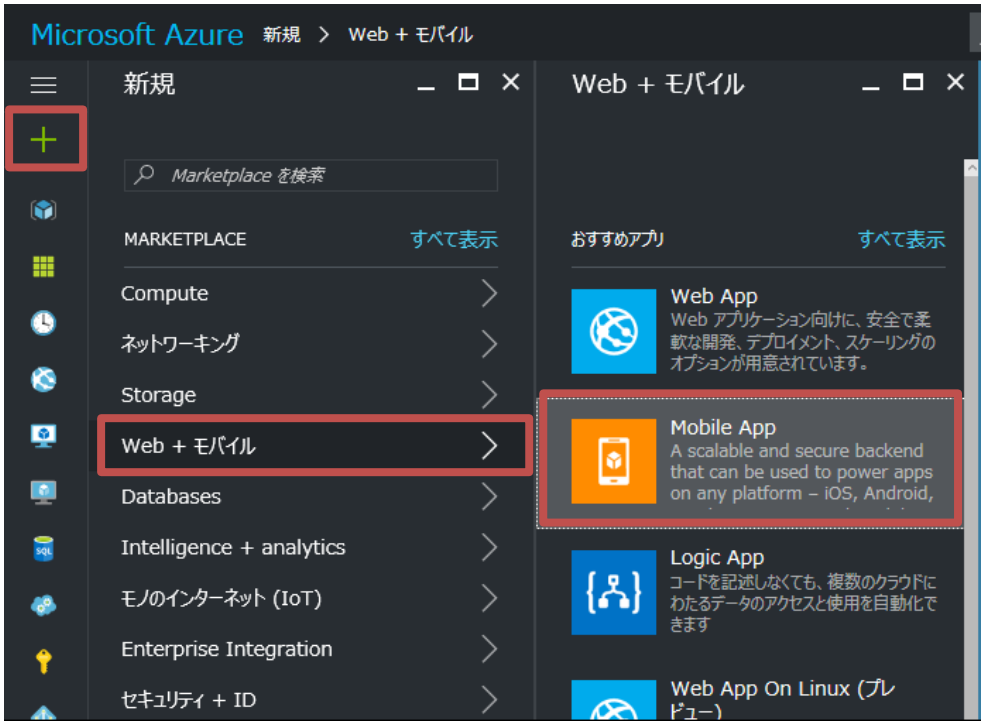
演習 1: Xamarin.Forms アプリと Azure Mobile App バックエンドを作成する

この演習では、Azure Mobile App バックエンドを使用して Xamarin.Forms モバイル アプリケーションにクラウド ベースのバックエンド サービスを追加する方法を示します。新しい Mobile App バックエンドと、アプリのデータを Azure に格納する簡単な Todo list Xamarin.Forms アプリの両方を作成します。

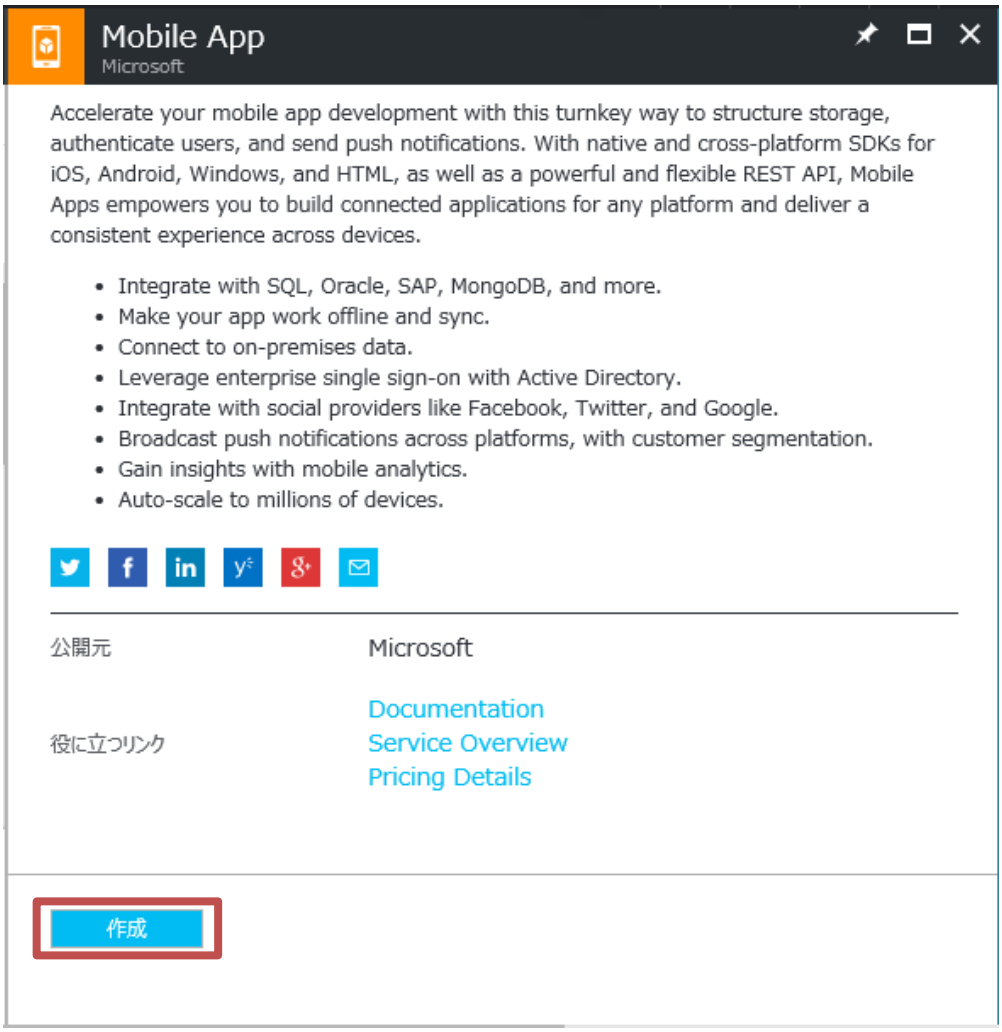
タスク 1 – 新しい Azure Mobile App バックエンドの作成

ここでは、新しいモバイル アプリ バックエンドを作成します。

1. まず、Azure ポータル (https://portal.azure.com/) にログインします。
2. 左上の [＋ 新規] をクリックします。
3. [新規] ブレードの [MARKETPLACE] から、[Web ＋ モバイル > Mobile App] を選択します。
   または、[Marketplace を検索] から [Mobile App] と入力し、Mobile App を選択します。



4. [Mobile App] ブレードが表示されたら、左下の [作成] をクリックします。



メモ

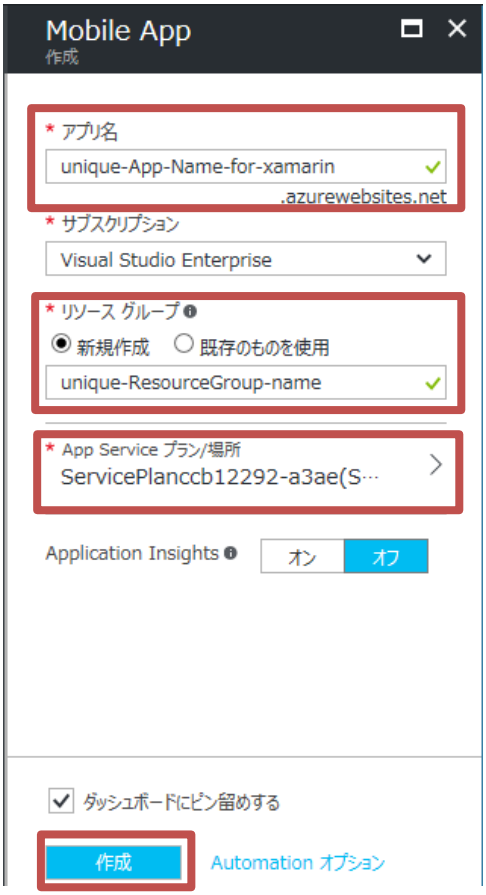
Marketplace を検索すると、[Mobile Apps Quickstart] が見つかりますが、こちらを利用しても問題ありません。

5. [Mobile App] の新規作成ブレードが表示されるので、下記の項目に対して入力を行います。

アプリ名	モバイル アプリ バックエンドの名前を入力します。 [入力されたアプリ名].azurewebsites.net という名称のドメイン名になります。
リソース グループ	Azure 上の様々なコンポーネントをまとめるコンテナー。既存のリソース グループを選択するか、新しく作成します。詳細は「Azure リソース マネージャーの概要」を参照してください。

App Service プラン　アプリをホストするために使用する物理リソースのコレクションです。アプリに関連付けられる場所、機能、コスト、コンピューティング リソースが決まります。App Services プランの詳細と、さまざまな価格レベルおよび目的の場所で新しいプランを作成する方法については、「Azure App Service プランの詳細な概要」を参照してください。

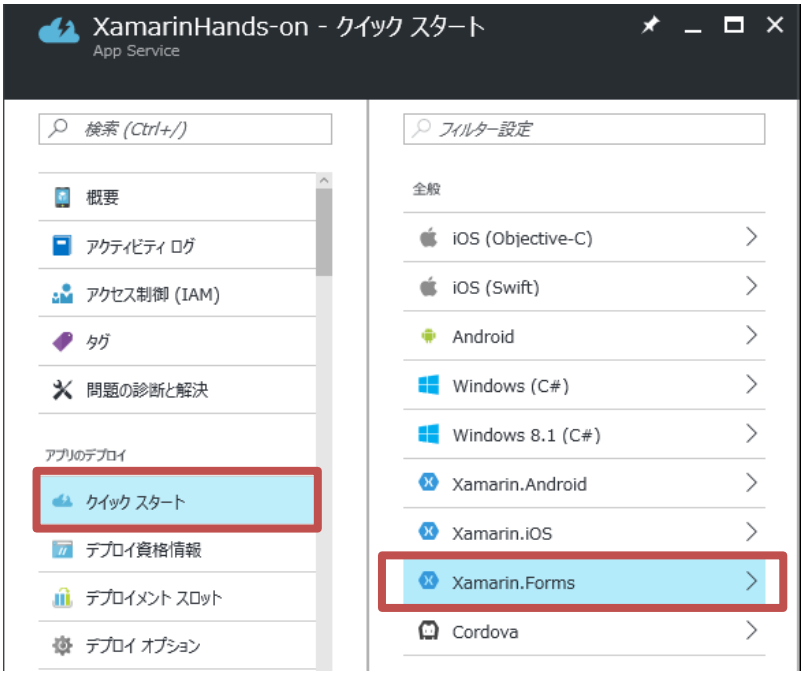
6. アプリ名、リソース グループ、App Service プランを決定したら、[作成] をクリックします。モバイル アプリ バックエンドが作成されます。



7. モバイル アプリ バックエンドが作成されるまで、しばらくお待ちください。

タスク 2 – サーバー プロジェクトの構成

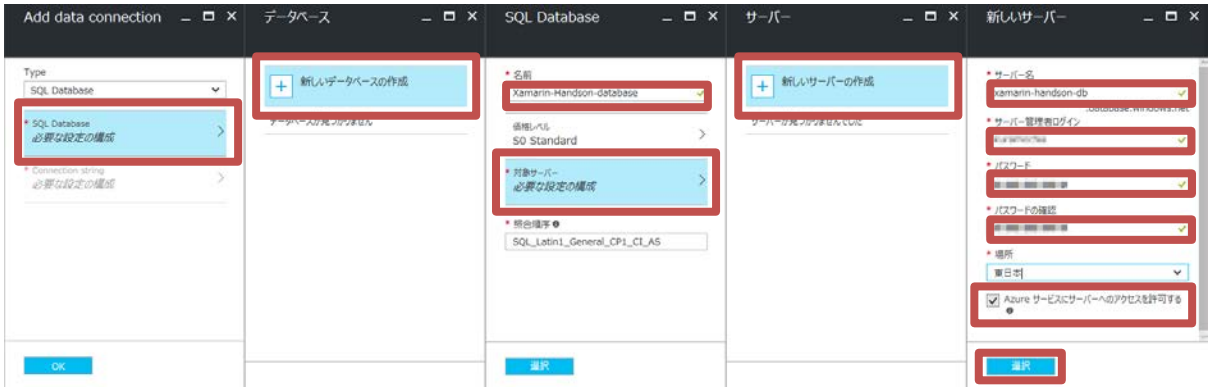
1. 新しいモバイル アプリ バックエンドの [設定] ブレードで、[クイック スタート]、[Xamarin.Forms] の順にクリックします。



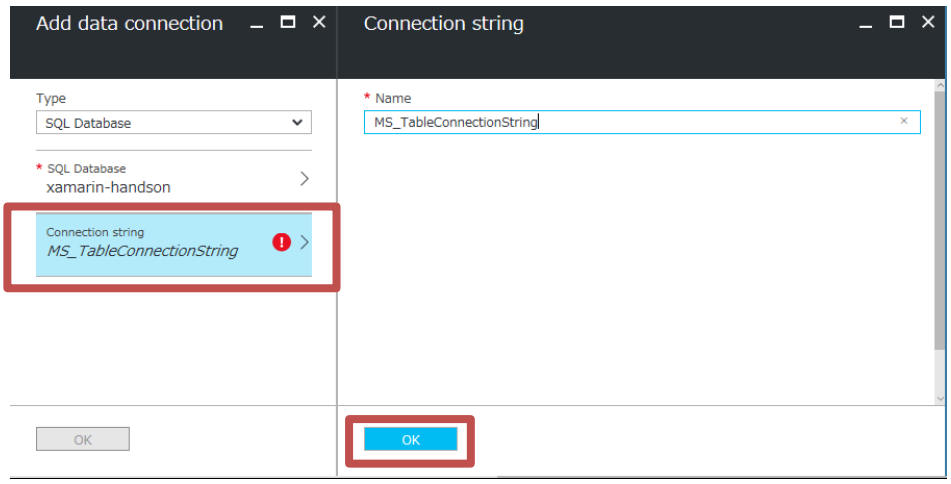
2. [1 データベースの接続] を選択し、新しいデータベースの作成に進みます。

3. [データ接続] ブレードが表示されるので、[＋追加] を選択します。

4. [データ接続の追加] ブレードで、[SQL Database] 、[新しいデータベースの作成] の順にクリックします。データベースの [名前] を入力し、価格レベルを選択して、[サーバー] をクリックします。この新しいデータベースは再利用できます。同じ場所に既にデータベースを所有している場合は、代わりに既存のデータベースを使用することもできます。別の場所にあるデータベースを使用することは、帯域幅コストと待機時間が増加するため、お勧めしません。
5. [新しいサーバー] ブレードで、[サーバー名] フィールドに一意のサーバー名を入力して、ログインとパスワードを指定し、[Azure サービスにサーバーへのアクセスを許可する] をオンにして、[OK] をクリックします。新しいデータベースが作成されます。



6. [データ接続の追加] ブレードに戻り、[接続文字列] をクリックし、既定値のまま、[OK] をクリックします。



データベースが正常にデプロイされるまで数分待ってから次の手順に進んでください。

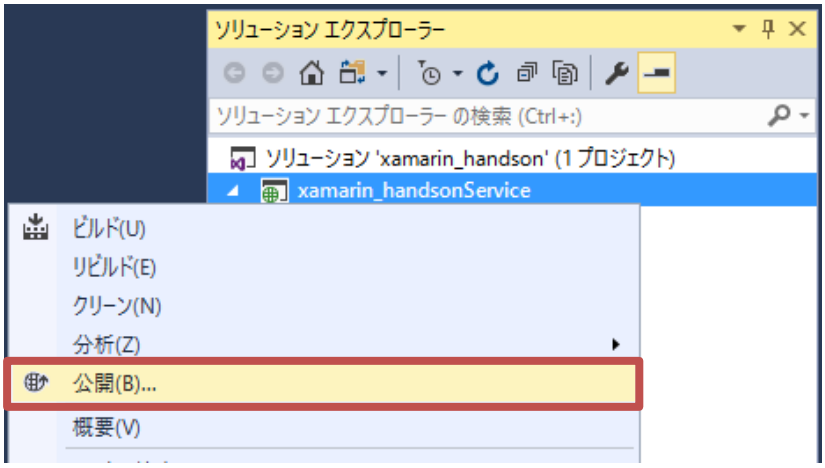
メモ

数分経っても状態が変わらない場合は、Web ブラウザの再読み込みを行ってください。

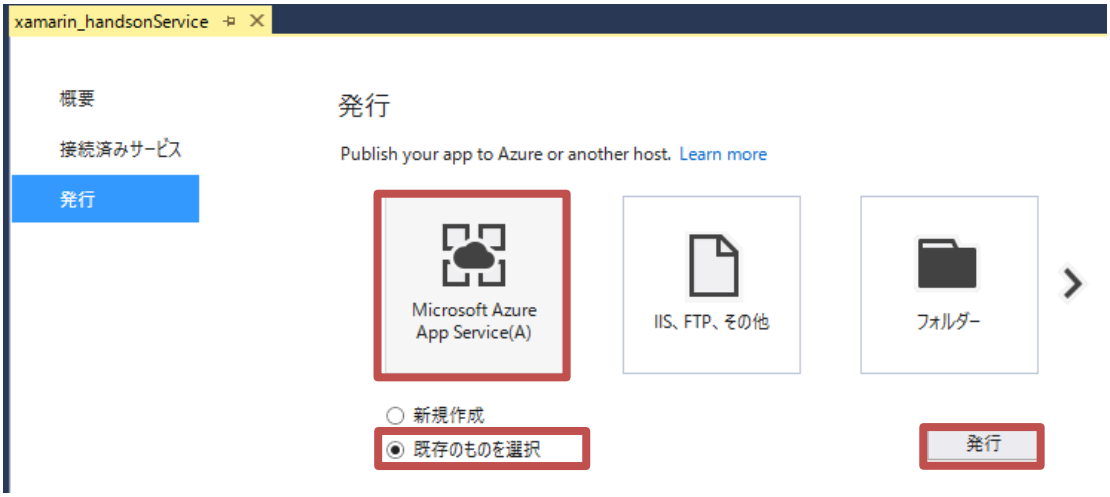
7. データベースが正常にデプロイされると、モバイル アプリ バックエンドとの接続が行われ、[Quick start] ブレードの[1 データベースの接続] の表示が、下記のようなチェックマークに変更されます。

8. [クイック スタート] ブレードに戻り、[2 テーブル API の作成] の [バックエンド 言語:] を [C#] に変更し、[ダウンロード] をクリックします。

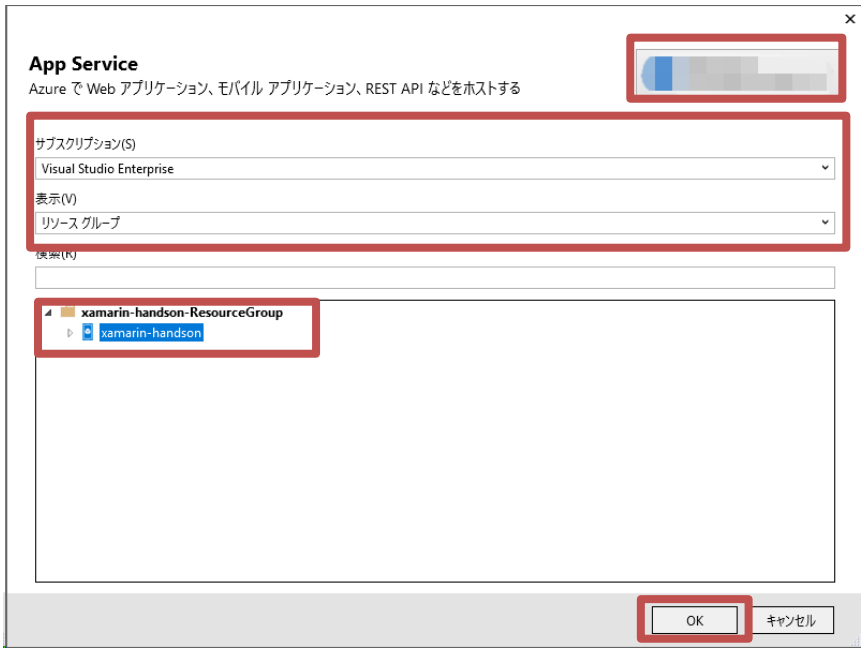
9. Zip 圧縮されたバックエンド プロジェクトのダウンロードが開始されますので、ローカル コンピューターに保存し、Zip ファイルを展開後、Visual Studio でソリューションを展開します。
10. バックエンド プロジェクトをビルドします。
11. [ソリューション エクスプローラー] より、バックエンド プロジェクトを右クリックし、[公開] をクリックします。



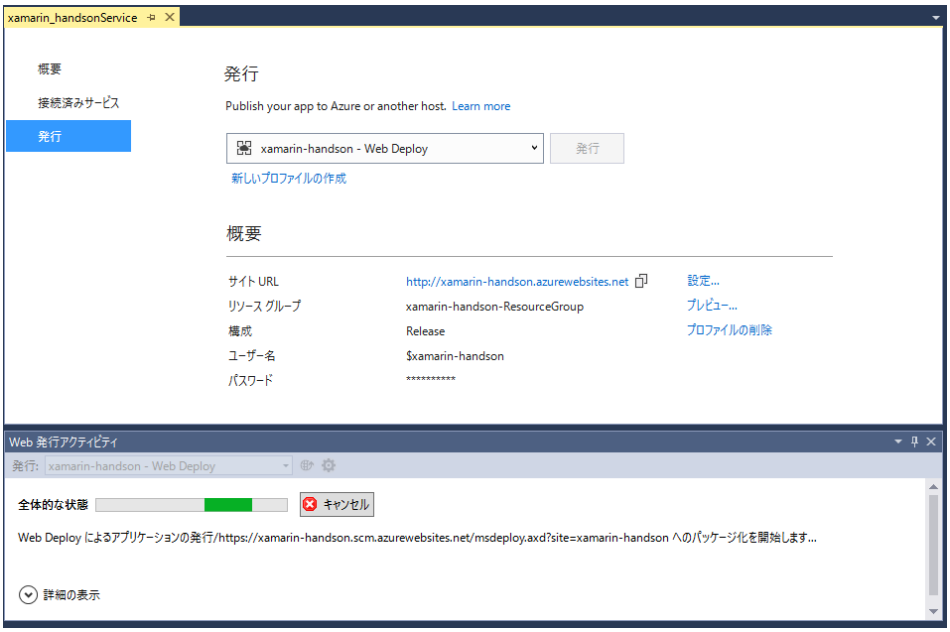
12. [発行] ウィンドウが表示されるので、[Microsoft Azure App Service] をクリックします。続いて、[既存のものを選択] を選択し、[発行] をクリックします。



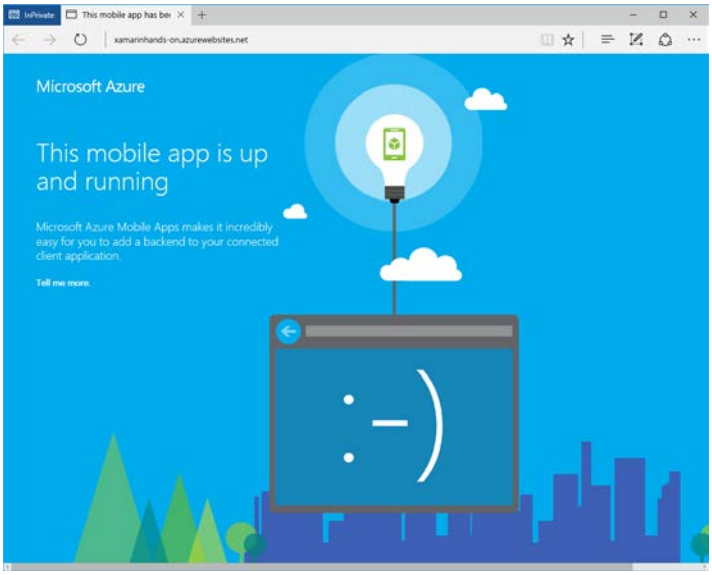
13. [App Service] ウィンドウで、Azure へのログイン アカウント、サブスクリプションを指定し、[表示] を [リソース グループ] に変更します。ウィンドウ下部に先ほど作成した、モバイル アプリ バックエンド が表示されますので、選択し、[OK] をクリックします。



14. モバイル バックエンド サービスが Release ビルドされ、Azure に発行されます。



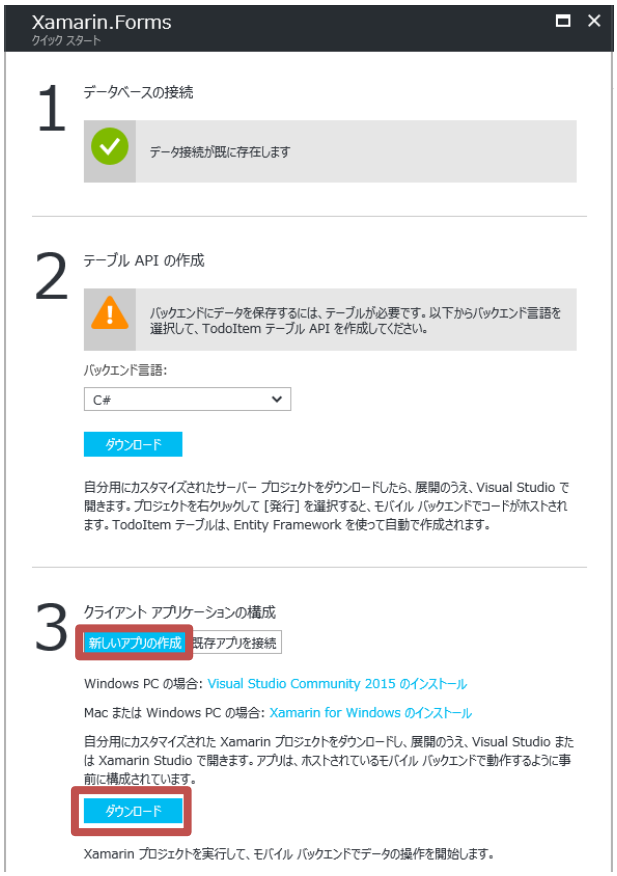
15. ブラウザが起動し、下記ページが表示されれば、バックエンド プロジェクトの Azure への発行が完了しました。



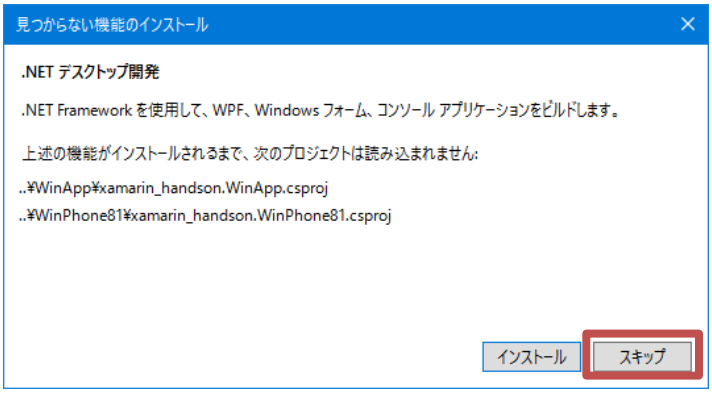
タスク 2 – Xamarin.Forms ソリューションのダウンロードと実行

続いて、クライアント側に移ります。Xamarin.Forms 用のソリューションは Microsoft Azure より実装済みの状態のソリューションをダウンロードすることができます。このタスクでは、ソリューションをダウンロードし、各プラットフォームでアプリケーションの実行を行います。

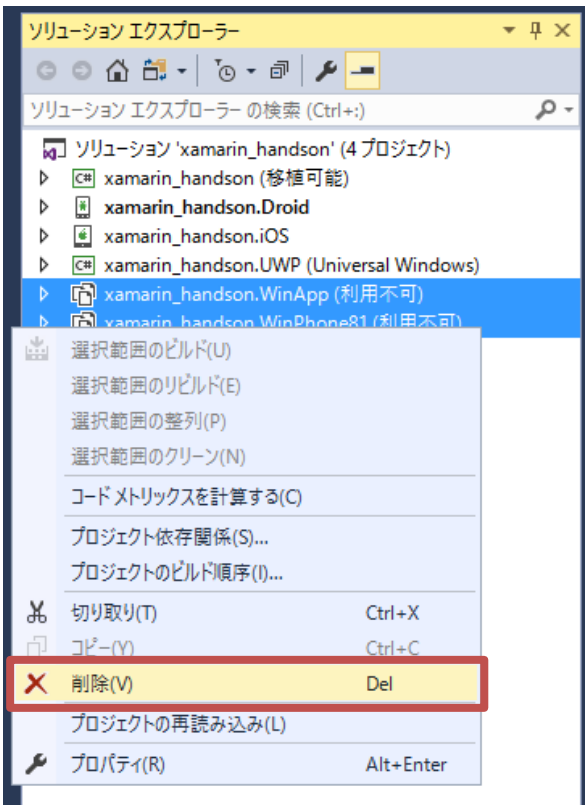
1. Azure 管理ポータルを表示し、[3 クライアント アプリケーションの構成] で [新しいアプリの作成] を選択し、[ダウンロード] ボタンをクリックします。これで、Xamarin.Forms のソリューション一式をダウンロードすることができます。



2. ダウンロードされた Zip ファイル (ファイル サイズはおよそ 300KB) を解凍して、.sln ファイルをダブルクリックし、Visual Studio を起動します。
3. ソリューションを起動すると、下記のようなウィンドウが表示されることがあります。*.WinApp.csproj および *.WinPhone81.csproj は、Windows 8.1 および Windows Phone 8.1 用のプロジェクトですので、今回はビルドする必要がありません。[スキップ] をクリックして先に進んでください。



4. [ソリューション エクスプローラー] に表示されているプロジェクトのうち、[*.WinApp] および [*.WinPhone81] という名称のプロジェクトが (利用不可) と表示されていることがあります。これは、Windows 8.1 および Windows Phone 8.1 用の SDK がインストールされていないためです。これらは今回の演習では利用しないため、Shift キーを押しながら、 [*.WinApp] および [*.WinPhone81] のプロジェクトを複数選択し、右クリックメニューの [削除] を選択して、プロジェクトを削除してください。



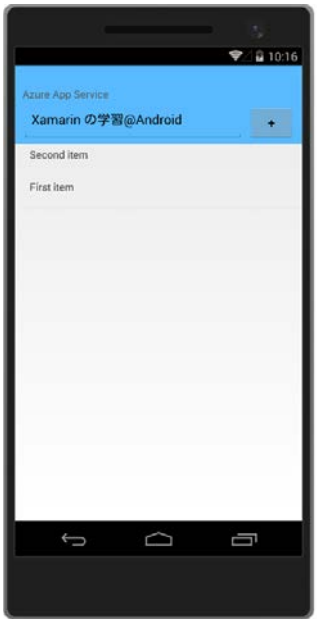
Android アプリケーションの実行

Visual Studio Emulator for Android で実行

1. Visual Studio 上部のメニューバーにある、3 つのコンボボックスを次のように設定します。

- ✓ ソリューション構成：**Debug**
- ✓ ソリューション プラットフォーム：**Any CPU**
- ✓ スタートアップ プロジェクト：**[AppName].Droid**

2. 次に、デバッグ ターゲットから [5" KitKat (4.4) XXHDPI Phone (Android 4.4 – API 19)] を選択し、デバッグを開始します。Android エミュレーターが起動し、アプリケーションが実行できていることを確認します。
3. Android エミュレーター上のアプリで、任意のテキストを入力し、[+] ボタンをクリックします。



これで、Azure でホストされている新しいモバイル アプリ バックエンドに POST 要求が送信されます。要求のデータは TodoItem テーブルに挿入されます。テーブルに格納された項目がモバイル アプリ バックエンドによって返され、データが一覧に表示されます。

メモ

ソリューションのポータブル クラス ライブラリ プロジェクトの TodoItemManager.cs C# ファイルに、モバイル アプリ バックエンドにアクセスするコードが表示されます。

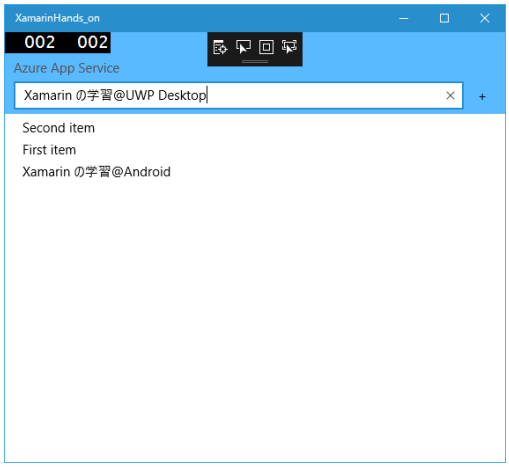
(オプション) Android 端末で実行

1. Visual Studio を実行している Windows 端末に、USB ケーブルで Android 端末を接続します。
2. 次に、デバッグ ターゲットに表示される Android 端末を選択し、デバッグを開始します。

UWP アプリケーションの実行

ローカル コンピューターで実行

1. Visual Studio 上部のメニューバーにある、3 つのコンボボックスを次のように設定します。
   - ✓ ソリューション構成：**Debug**
   - ✓ ソリューション プラットフォーム：**x86 または x64**
   - ✓ スタートアップ プロジェクト：**[AppName].UWP (Universal Windows)**
2. 次に、デバッグ ターゲットから [ローカル コンピューター] を選択し、デバッグを開始します。



Windows 10 Mobile Emulator で実行

1. Visual Studio 上部のメニューバーにある、3 つのコンボボックスを次のように設定します。
   - ✓ ソリューション構成：**Debug**
   - ✓ ソリューション プラットフォーム：**x86**
   - ✓ スタートアップ プロジェクト： **[AppName].UWP (Universal Windows)**
2. 次に、デバッグ ターゲットから [Mobile Emulator 10.0.15063.0 WVGA 4 inch 1GB] を選択し、デバッグを開始します。

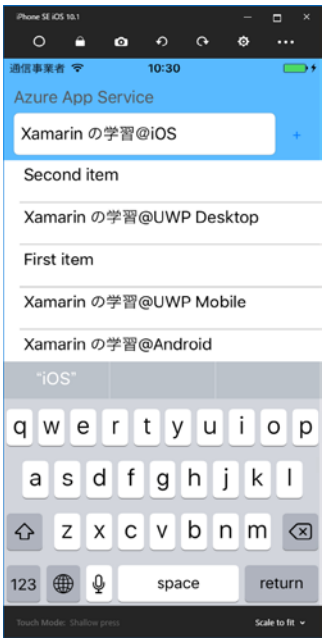
(オプション) Windows 10 Mobile 端末で実行

1. Visual Studio 上部のメニューバーにある、3 つのコンボボックスを次のように設定します。
   - ✓ ソリューション構成：**Debug**
   - ✓ ソリューション プラットフォーム：**ARM**
   - ✓ スタートアップ プロジェクト：**[AppName].UWP**
2. 次に、デバッグ ターゲットから [Device] を選択し、デバッグを開始します。

(オプション) iOS アプリケーションの実行

iOS Simulator (または、iOS Simulator for Windows) で実行

1. Visual Studio 上部のメニューバーにある、3 つのコンボボックスを次のように設定します。
   - ✓ ソリューション構成：**Debug**
   - ✓ ソリューション プラットフォーム：**iPhoneSimulator**
   - ✓ スタートアップ プロジェクト：**[AppName].iOS**
2. 次に、デバッグ ターゲットから [iPhone SE iOS 10.3] を選択し、デバッグを開始します。



メモ

iOS Simulator for Windows は Visual Studio 2017 Enterprise エディションのみ、利用可能です。Community および Professional エディションの場合は、macOS 上の Simulator を利用できます。

iOS 端末で実行

1. Visual Studio 上部のメニューバーにある、3 つのコンボボックスを次のように設定します。
   - ✓ ソリューション構成：**Debug**

- ✓ ソリューション プラットフォーム：**iPhone**
- ✓ スタートアップ プロジェクト： **[AppName].iOS**

2. 次に、デバッグ ターゲットから接続した iOS 端末名を選択し、デバッグを開始します。

演習 2: Xamarin.Forms アプリへ認証を追加する

この演習では、クライアント アプリケーションから App Service モバイル アプリのユーザーを認証する方法について説明します。この演習では、App Service でサポートされている ID プロバイダーを使用して、演習 1 で作成したプロジェクトに認証を追加します。モバイル アプリによって正常に認証、承認されると、ユーザー ID 値が表示され、制限付きのテーブル データにアクセスできます。

前提条件

演習 1 を完了している必要があります。

アプリケーションを認証に登録し、App Services を構成する

最初に、ID プロバイダーのサイトでアプリを登録する必要があります。その後、プロバイダーによって生成された資格情報をモバイル アプリ バックエンドに設定します。

この演習では、ID プロバイダーとして Twitter を利用しますが、App Services では下記の認証に対応しています。各 ID プロバイダーを利用したい場合は、それぞれのリンク先の情報に従って、ID プロバイダーの構成を行ってください。

なお、各プロバイダーを利用するためには、アカウント登録が必要になります。

- Azure Active Directory
- Facebook
- Google
- Microsoft
- Twitter

> **メモ**
>
> Google ログインでの認証を行った場合、Android および iOS で実行すると、"Error : disallowed_useragent" というエラーが発生します。これは、Google 側が アプリ組み込みの WebView による認証をブロックしたためです。Azure Mobile Apps の認証画面の表示を行う

Microsoft.Azure.Mobile.Client コンポーネントは内部的に Xamarin.Auth のソースコードを利用しており、Xamarin.Auth 側での対応を行っています。(2017 年 4 月現在)

詳細は下記をご参照ください。

Google Developers Blog
Modernizing OAuth interactions in Native Apps for Better Usability and Security

Azure-mobile-apps-client (Github issue)
error : 403 disallowed_useragent while authenticating with google using azure services in iOS #263

Xamarin.auth (Github issue)
Google Auth Error:disallowed_useragent #137

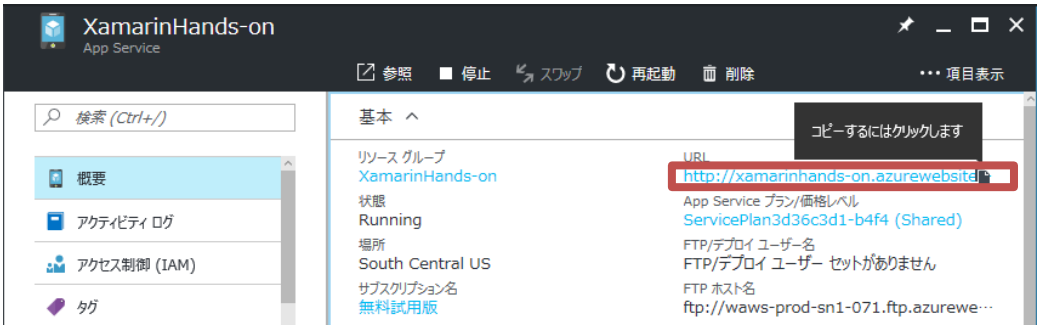
Twitter ログインを使用するように App Service アプリケーションを構成する

このトピックでは、認証プロバイダーとして Twitter を使用するように Azure App Services を構成する方法を示します。

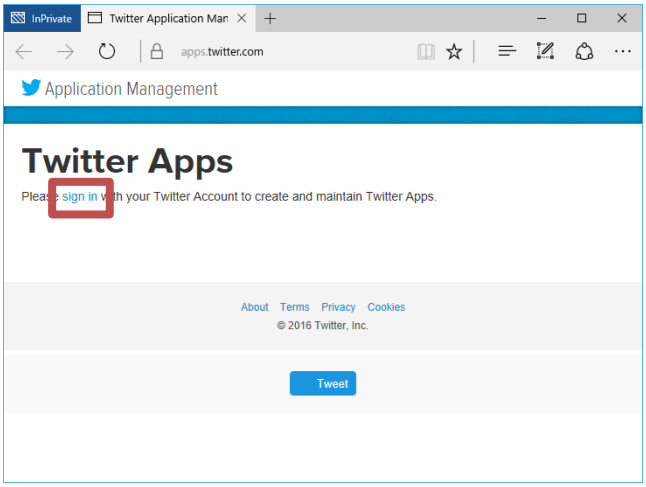
このトピックの手順を完了するには、電子メール アドレスと電話番号を検証済みの Twitter アカウントが必要になります。新しい Twitter アカウントを作成するには、twitter.com にアクセスしてください。

Twitter にアプリケーションを登録する

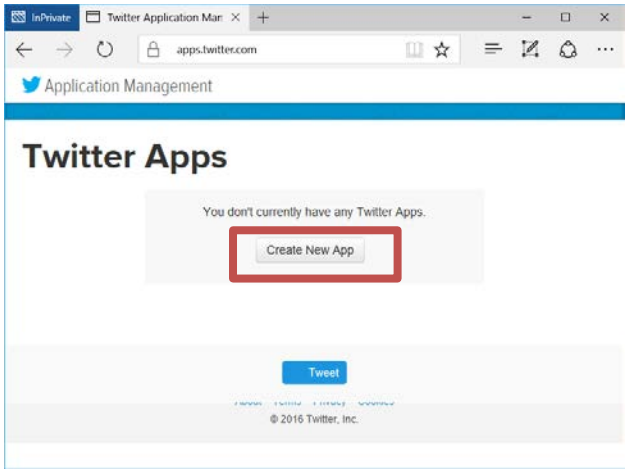
1. Azure ポータル にログオンし、アプリケーションに移動します。[URL] をコピーします。この URL は、Twitter アプリの構成で使用します。



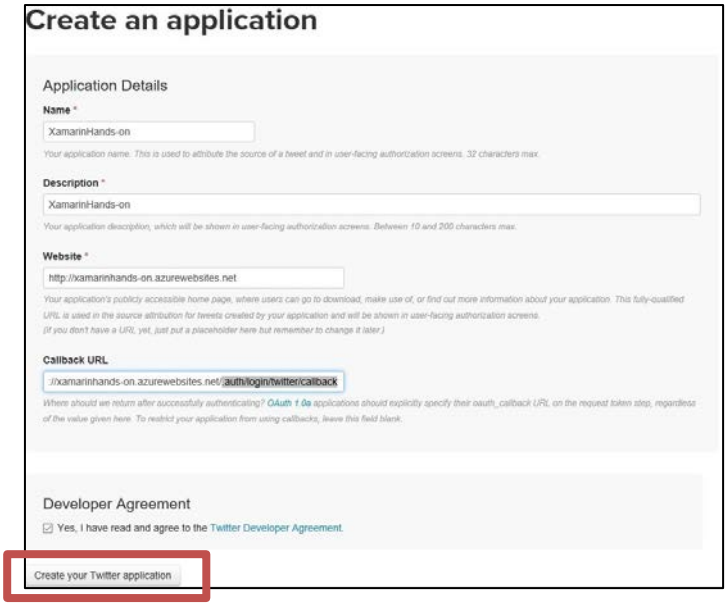
2. Twitter Developers の Web サイトに移動し、Twitter アカウント資格情報でサインインします。



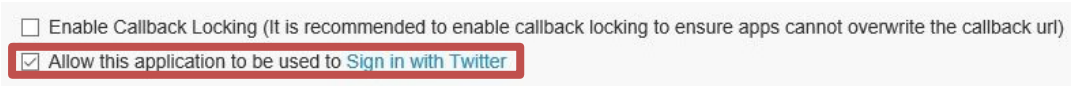
3. Twitter アカウント資格情報でサインインできましたら、[Create New App] をクリックします。



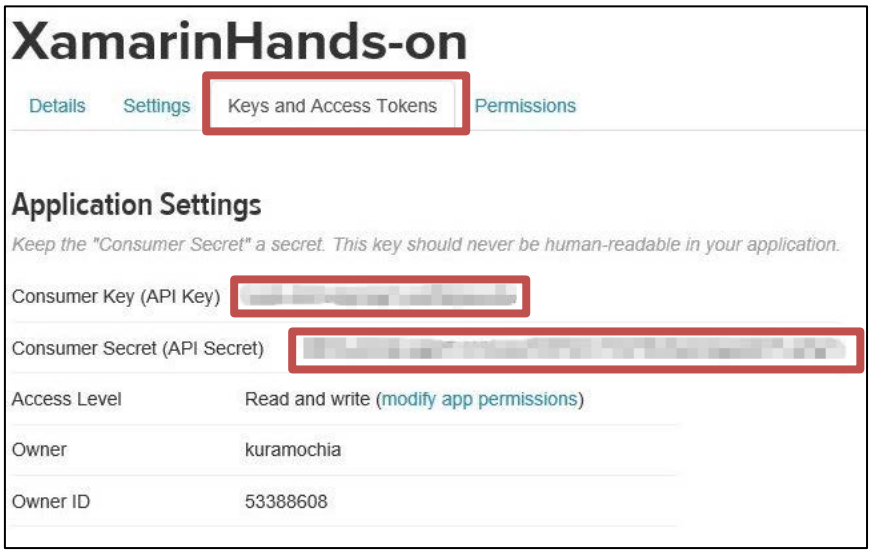
4. [Create an application] の各入力項目を、次のように入力します。
   - [Name]、[Description] を入力します。
   - [Website] に、先ほどコピーした URL を貼り付けます。
   - [Callback URL] は、下記のルールで作成した URL を貼り付けます。
     - 先ほどコピーした URL を http:// から https:// に変更
     - URL に パス [**/.auth/login/twitter/callback**] を追加
       例えば、https://contoso.azurewebsites.net/**.auth/login/twitter/callback** のように入力します。
5. [Developer Agreement] にチェックを入れ、[Create your Twitter application] をクリックします。



6. [Setting] タブをクリックし、[Allow this application to be used to sign in with Twitter] チェック ボックスをオンにして、[Update Settings] をクリックします。



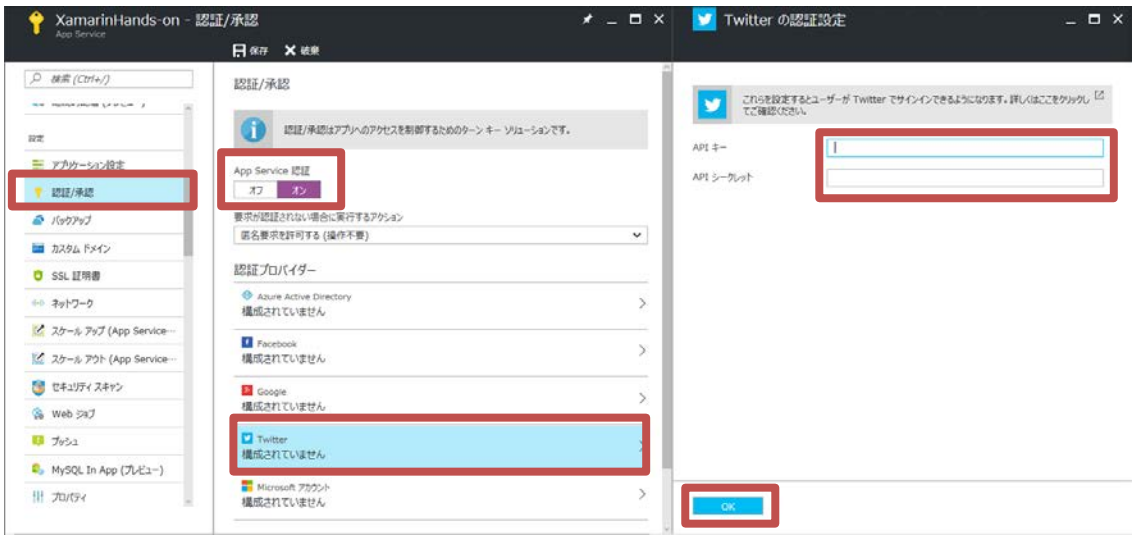
7. [Keys and Access Tokens] タブをクリックします。[Consumer Key (API Key)] と [Consumer secret (API Secret)] の値を書き留めます。



アプリケーションに Twitter 情報を追加する

1. Azure ポータル に戻り、アプリケーションに移動します。[認証/承認] をクリックします。[認証/承認] ブレードが表示されたら、[App Service 認証] を [オン] に変更し、[認証プロバイダー] の [Twitter] をクリックします。[Twitter の認証設定] ブレードの [API キー] に先ほどの [Consumer Key (API Key)]を、

[API シークレット] に先ほどの [Consumer Secret (API Secret)] をそれぞれ入力し、[OK] をクリックします。



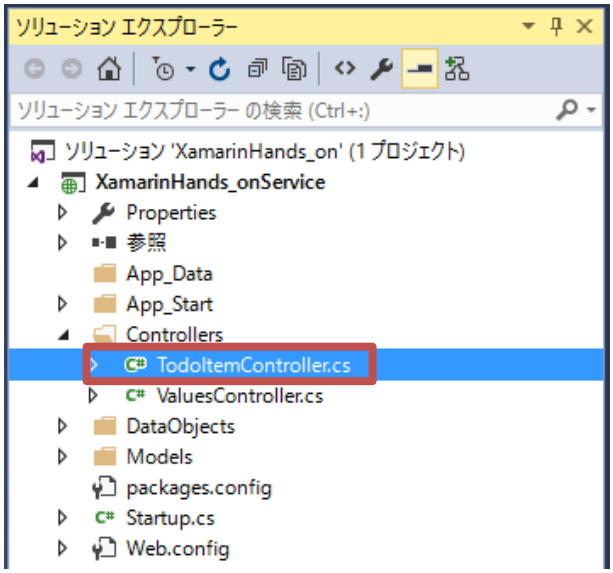
2. (省略可能) Twitter によって認証されたユーザーしかサイトにアクセスできないように制限するには、[要求が認証されていないときに実行するアクション] を [Twitter] に設定します。この場合、要求はすべて認証される必要があり、認証されていない要求はすべて認証のために Twitter にリダイレクトされます。

3. [認証/承認] ブレード上部の [保存] をクリックし、設定した内容を保存します。

アクセス許可を、認証されたユーザーだけに制限する

既定では、モバイル アプリ バックエンドの API は匿名で呼び出すことができます。次に、認証されたクライアントのみにアクセスを制限する必要があります。

1. サーバー プロジェクトで、[Controller > TodoItemController.cs] の順に移動します。



2. 次のように、[Authorize] 属性を TodoItemController クラスに追加します。アクセスを特定のメソッドのみに制限するには、この属性を、クラスではなく、そのメソッドのみに適用するだけです。

更新前

TodoItemController.cs

```
public class TodoItemController : TableController<TodoItem>
{
```

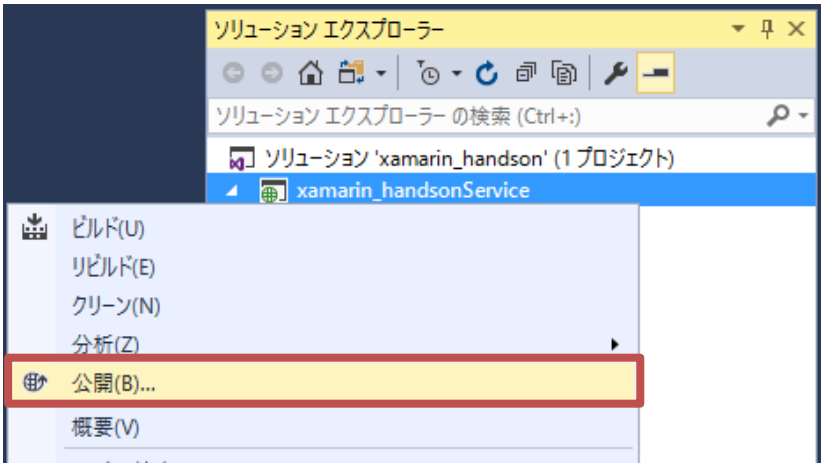
更新後

TodoItemController.cs

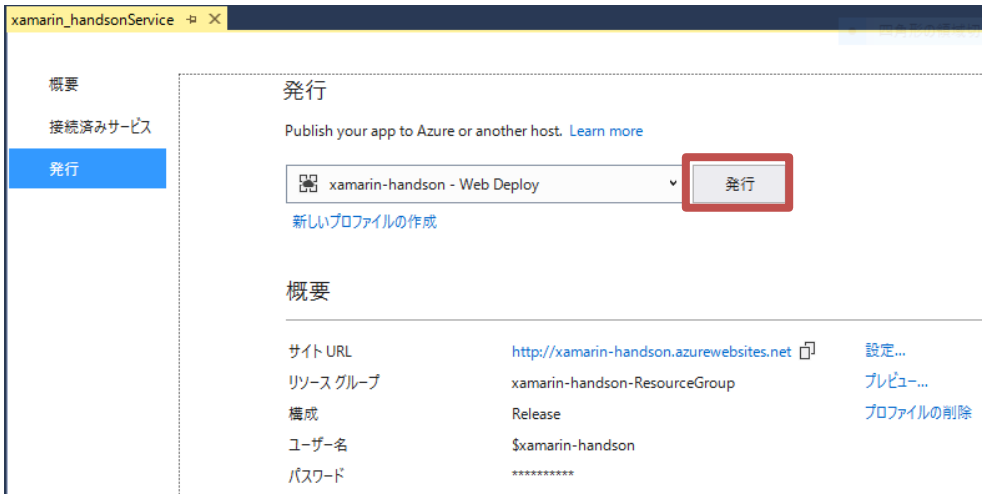
```
[Authorize]
public class TodoItemController : TableController<TodoItem>
{
```

Ctrl+S キーを押して、保存をしてください。

3. サーバー プロジェクトを右クリックし、メニューから [公開] をクリックします。



4. [発行] ウィンドウが表示されるので、[発行] ボタンをクリックします。



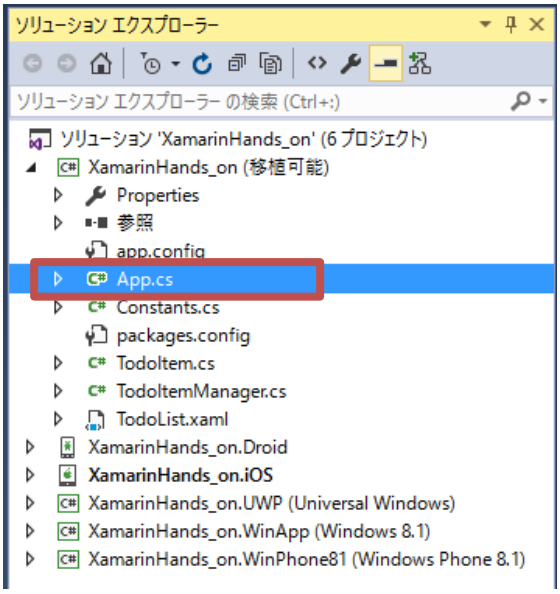
ポータブル クラス ライブラリに認証を追加する

Mobile Apps では、App Service 認証を使用したユーザーのサインインに、MobileServiceClient の LoginAsync 拡張メソッドを使います。このサンプルでは、アプリにプロバイダーのサインイン インターフェイスが表示される、サーバー側管理認証フローを使用します。詳細については、「サーバー側管理認証」を参照してください。運用アプリのユーザー エクスペリエンスを向上させるためには、代わりにクライアント側管理認証を使用することを検討してください。

Xamarin.Forms プロジェクトで認証するには、アプリのポータブル クラス ライブラリに IAuthenticate インターフェイスを定義します。また、ポータブル クラス ライブラリで定義したユーザー インターフェイスを更新して、認証を開始するためにユーザーがクリックする [サインイン] ボタンを追加します。認証が成功すると、モバイル アプリ バックエンドからデータが読み込まれます。

アプリでサポートされているプラットフォームごとに IAuthenticate インターフェイスを実装する必要があります。

1. Visual Studio で、ポータブル クラス ライブラリ プロジェクトである **[** xxxxxx (移植可能) **]** というプロジェクトから App.cs を開きます。



2. using ステートメントを追加します。また、IAuthenticate インターフェイス定義を、App クラス定義の直前に追加します。(namespace は、演習 1 で作成した モバイル アプリ バックエンド の名前によって異なります。)

更新前

App.cs

```
using System;

using Xamarin.Forms;

namespace XamarinHands_on
{
   public class App : Application
   {
```

更新後

App.cs

```
using System;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace XamarinHands_on
{
    public interface IAuthenticate
    {
        Task<bool> AuthenticateAsync();
    }

    public class App : Application
    {
```

3. インターフェイスをプラットフォーム固有の実装で初期化するように、次の静的メンバーを App クラスに追加します。

更新前

App.cs

```
public class App : Application
{
    public App ()
    {
     // The root page of your application
     MainPage = new TodoList();
}
```

更新後

App.cs

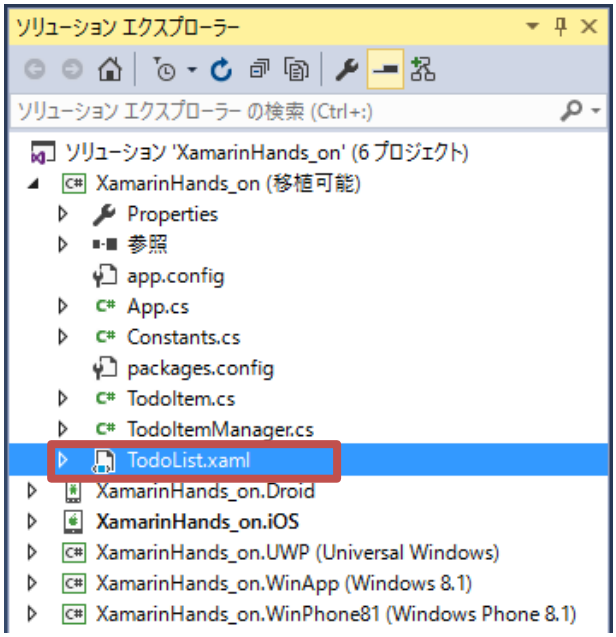
```
public class App : Application
{
    public static IAuthenticate Authenticator { get; private set; }

    public static void Init(IAuthenticate authenticator)
    {
        Authenticator = authenticator;
    }

    public App ()
{
     // The root page of your application
     MainPage = new TodoList();
}
```

Ctrl+S キーを押して、保存をしてください。

4. ポータブル クラス ライブラリ プロジェクトから TodoList.xaml を開きます。



5. 既存のボタンの後ろに、buttonsPanel レイアウト要素の次の Button 要素を追加します。

更新前

TodoList.xaml

```
        <StackLayout x:Name="buttonsPanel" Grid.Column="1"
Orientation="Horizontal" HorizontalOptions="StartAndExpand">
          <Button Text="+"
                  MinimumHeightRequest="30"
                  Clicked="OnAdd" />
        </StackLayout>
```

更新後

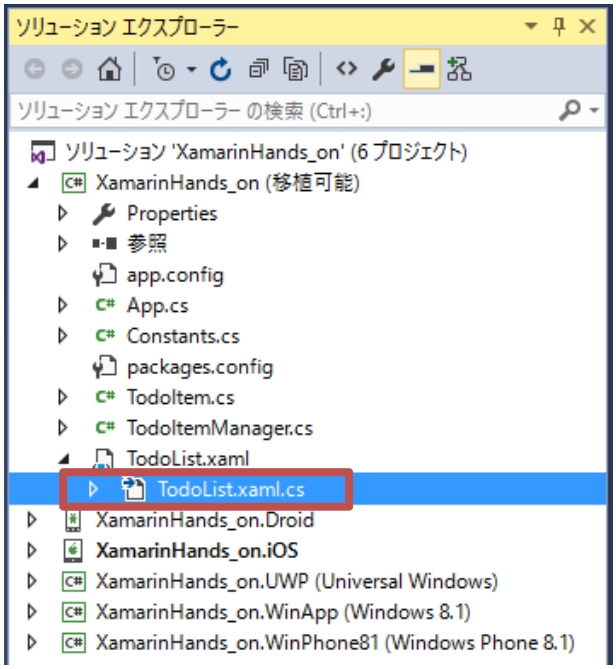
TodoList.xaml

```
        <StackLayout x:Name="buttonsPanel" Grid.Column="1"
Orientation="Horizontal" HorizontalOptions="StartAndExpand">
          <Button Text="+"
                  MinimumHeightRequest="30"
                  Clicked="OnAdd" />
          <Button x:Name="loginButton" Text="Sign-in"
MinimumHeightRequest="30"
                  Clicked="loginButton_Clicked"/>
        </StackLayout>
```

このボタンは、モバイル アプリ バックエンドによるサーバー側管理認証をトリガーします。

Ctrl+S キーを押して、保存をしてください。

6. ポータブル クラス ライブラリ プロジェクトから TodoList.xaml.cs を開きます。



7. TodoList クラスに次のフィールドを追加します。

更新前

TodoList.xaml.cs

```
using System;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace XamarinHands_on
{
    public partial class TodoList : ContentPage
    {
        TodoItemManager manager;
```

更新後

TodoList.xaml.cs

```
using System;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace XamarinHands_on
{
    public partial class TodoList : ContentPage
    {
        // ユーザーが認証されているかどうかを追跡します。
        bool authenticated = false;

        TodoItemManager manager;
```

8. 続いて、TodoList クラスの OnAppearing メソッドを次のコードに置き換えます。

更新前

TodoList.xaml.cs

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

    // Set syncItems to true in order to synchronize the data on startup when
running in offline mode
    await RefreshItems(true, syncItems: true);
}
```

更新後

TodoList.xaml.cs

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

    // 認証された時のみ、項目を更新します。
    if (authenticated == true)
    {
        // オフライン モード で実行している場合、
        // 起動時にデータを同期するために、syncItems に true を設定します。
        await RefreshItems(true, syncItems: false);

        // サインインボタンを非表示にします。
        this.loginButton.IsVisible = false;
    }
}
```

この実装により、ユーザー認証の完了後にのみ、データがサービスによって更新されるようになります。

9. 次に、Clicked イベントの次のハンドラーを TodoList クラスに追加します。

更新前

TodoList.xaml.cs

```
public partial class TodoList : ContentPage
{
    // ユーザーが認証されているかどうかを追跡します。
    bool authenticated = false;

    TodoItemManager manager;

    public TodoList()
```

```
    {
```

更新後

TodoList.xaml.cs

```
public partial class TodoList : ContentPage
{
    // ユーザーが認証されているかどうかを追跡します。
    bool authenticated = false;

    TodoItemManager manager;

    async void loginButton_Clicked(object sender, EventArgs e)
    {
        if (App.Authenticator != null)
            authenticated = await App.Authenticator.AuthenticateAsync();

        // 起動時にオフライン データを同期する場合は、syncItems に true を設定しま
す。
        if (authenticated == true)
            await RefreshItems(true, syncItems: false);
    }

    public TodoList()
    {
```

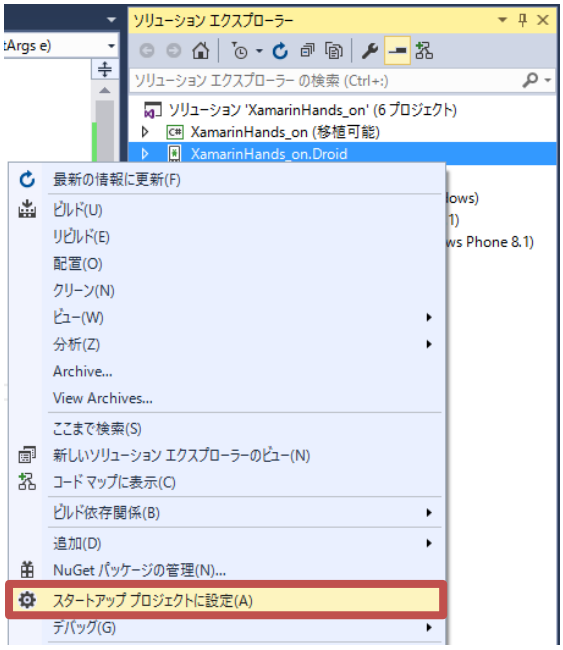
Ctrl+S キーを押して、保存をしてください。

10. ポータブル クラス ライブラリ プロジェクトをリビルドしてエラーがないことを確認します。

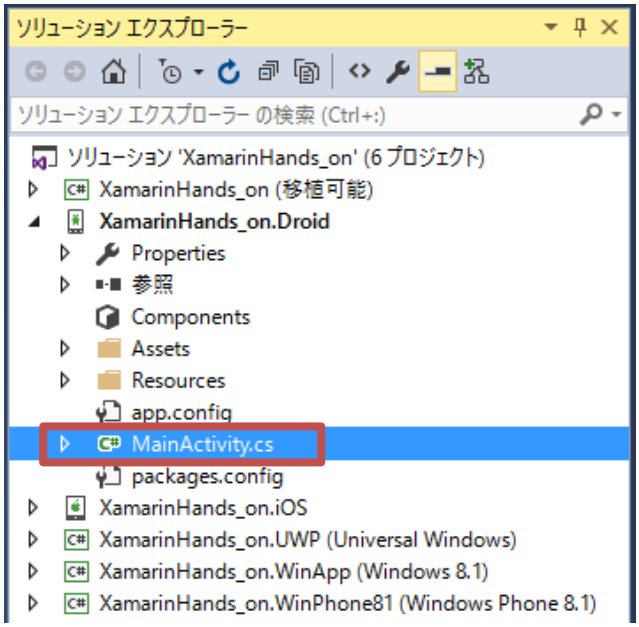
Android アプリに認証を追加する

このセクションでは、Android アプリ プロジェクト内に IAuthenticate インターフェイスを実装する方法について説明します。

1. Android プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。



2. F5 キーを押してデバッガーでプロジェクトを開始し、アプリの開始後に、状態コード 401 のハンドルされない例外 (許可されていません) が発生することを確認します。これは、バックエンドへのアクセスが承認済みのユーザーのみに制限されているために発生します。
3. Android プロジェクトの MainActivity.cs を開きます。



4. 次の using ステートメントを追加します。

更新前

MainActivity.cs

```
using System;
```

```
using Android.App;
using Android.Content;
using Android.Content.PM;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
```

更新後

MainActivity.cs

```
using System;
using System.Threading.Tasks;
using Microsoft.WindowsAzure.MobileServices;

using Android.App;
using Android.Content;
using Android.Content.PM;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
```

5. 次のように、MainActivity クラスを更新して IAuthenticate インターフェイスを実装します。

更新前

MainActivity.cs

```
public class MainActivity :
global::Xamarin.Forms.Platform.Android.FormsApplicationActivity
```

更新後

MainActivity.cs

```
public class MainActivity :
global::Xamarin.Forms.Platform.Android.FormsApplicationActivity, IAuthenticate
```

6. 次のように、MobileServiceUser フィールドのほか、IAuthenticate インターフェイスに必要な AuthenticateAsync メソッドを追加して、MainActivity クラスを更新します。

更新前

MainActivity.cs

```
public class MainActivity :
global::Xamarin.Forms.Platform.Android.FormsApplicationActivity, IAuthenticate
{
```

```
  protected override void OnCreate (Bundle bundle)
  {
```

更新後

MainActivity.cs

```
public class MainActivity :
global::Xamarin.Forms.Platform.Android.FormsApplicationActivity, IAuthenticate
{
    // 認証されたユーザーの定義
    private MobileServiceUser user;

    public async Task<bool> AuthenticateAsync()
    {
        var success = false;
        var message = string.Empty;
        try
        {
            // Twitterで サインインを行います。
            user = await
TodoItemManager.DefaultManager.CurrentClient.LoginAsync(
                this,
                MobileServiceAuthenticationProvider.Twitter);
            if (user != null)
            {
                message = message = $"you are now signed-in as
{user.UserId}.";
                success = true;
            }
        }
        catch (Exception ex)
        {
            message = ex.Message;
        }

        // 成功または失敗を表示します。
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.SetMessage(message);
        builder.SetTitle("Sign-in result");
        builder.Create().Show();

        return success;
    }

    protected override void OnCreate (Bundle bundle)
    {
```

Twitter 以外の ID プロバイダーを使用している場合、MobileServiceAuthenticationProvider の Twitter となっているコードに、各プロバイダーの値を選択してください。

7. LoadApplication() の呼び出しの前にある MainActivity クラスの OnCreate メソッドに次のコードを追加します。

更新前

MainActivity.cs

```
protected override void OnCreate (Bundle bundle)
{
   base.OnCreate (bundle);

   // Initialize Azure Mobile Apps
   Microsoft.WindowsAzure.MobileServices.CurrentPlatform.Init();

   // Initialize Xamarin Forms
   global::Xamarin.Forms.Forms.Init (this, bundle);

   // Load the main application
   LoadApplication (new App ());
}
```

更新後

MainActivity.cs

```
protected override void OnCreate (Bundle bundle)
{
   base.OnCreate (bundle);

   // Initialize Azure Mobile Apps
   Microsoft.WindowsAzure.MobileServices.CurrentPlatform.Init();

   // Initialize Xamarin Forms
   global::Xamarin.Forms.Forms.Init (this, bundle);

      // アプリがロードされる前に、authenticator を初期化します。
      App.Init((IAuthenticate)this);

   // Load the main application
   LoadApplication (new App ());
}
```

これにより、アプリの読み込み前に Authenticator が初期化されるようになります。

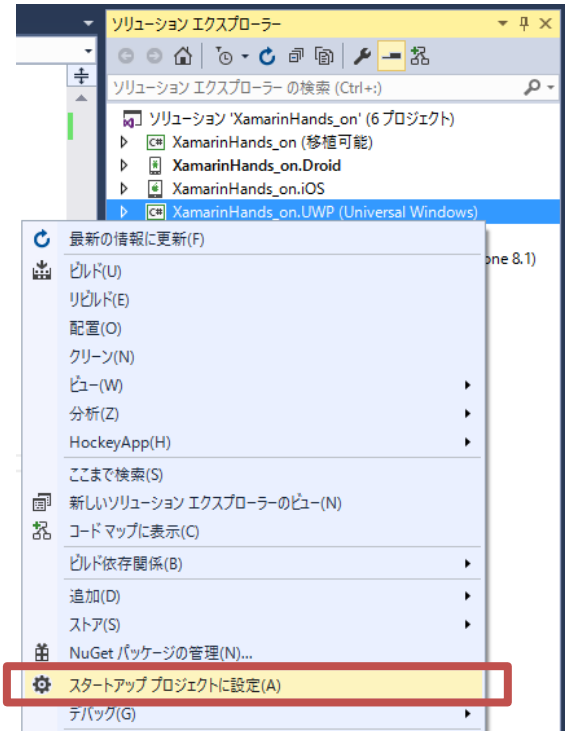
Ctrl+S キーを押して、保存をしてください。

8. アプリをリビルドして実行します。その後で、選択した認証プロバイダーを使用してサインインし、認証されたユーザーとしてデータにアクセスできることを確認します。

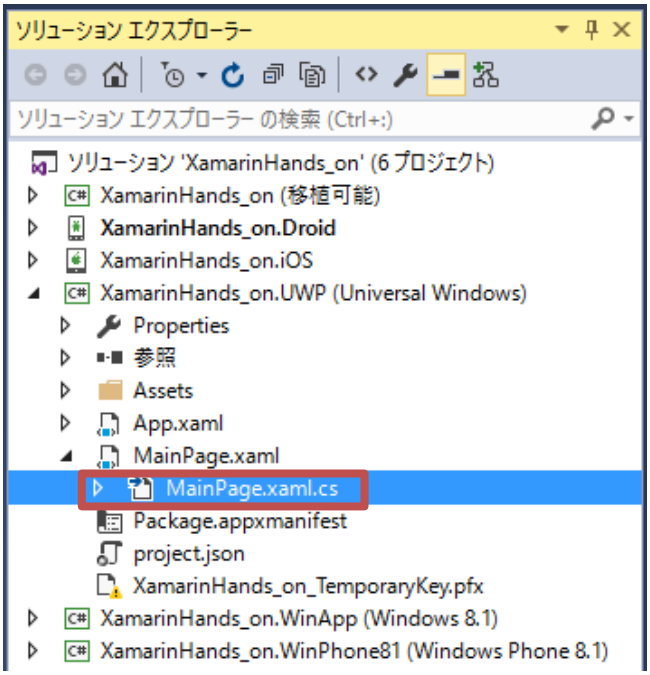
UWP アプリに認証を追加する

このセクションでは、ユニバーサル Windows プラットフォーム (UWP) プロジェクト内に IAuthenticate インターフェイスを実装する方法について説明します。

1. UWP プロジェクトを右クリックし、**[スタートアップ プロジェクトに設定]** をクリックします。



2. F5 キーを押してデバッガーでプロジェクトを開始し、アプリの開始後に、状態コード 401 のハンドルされない例外 (許可されていません) が発生することを確認します。これは、バックエンドへのアクセスが承認済みのユーザーのみに制限されているために発生します。
3. UWP アプリ プロジェクトの MainPage.xaml.cs を開きます。



4. 続いて、MainPage.xaml.cs に次の using ステートメントを追加します。

更新前

MainPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;
```

更新後

MainPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation;
```

```
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

using Microsoft.WindowsAzure.MobileServices;
using System.Threading.Tasks;
using Windows.UI.Popups;
using <your_Portable_Class_Library_namespace>;
```

<your_Portable_Class_Library_namespace> を、ポータブル クラス ライブラリの名前空間に置き換えます。

5. 次のように、MainPage クラスを更新して IAuthenticate インターフェイスを実装します。

更新前

MainPage.xaml.cs

```
public sealed partial class MainPage
```

更新後

MainPage.xaml.cs

```
public sealed partial class MainPage : IAuthenticate
```

6. 次のように、MobileServiceUser フィールドのほか、IAuthenticate インターフェイスに必要な AuthenticateAsync メソッドを追加して、MainPage クラスを更新します。

更新前

MainPage.xaml.cs

```
public sealed partial class MainPage : IAuthenticate
{
    public MainPage()
    {
        this.InitializeComponent();

        LoadApplication(new <your_Portable_Class_Library_namespace>.App());
```

```
    }
}
```

更新後

MainPage.xaml.cs

```
public sealed partial class MainPage : IAuthenticate
{
    // 認証されたユーザーの定義
    private MobileServiceUser user;

    public async Task<bool> AuthenticateAsync()
    {
        string message = string.Empty;
        var success = false;

        try
        {
            // Twitterで サインインを行います。
            if (user == null)
            {
                user = await TodoItemManager.DefaultManager.CurrentClient
                    .LoginAsync(MobileServiceAuthenticationProvider.Twitter);
                if (user != null)
                {
                    success = true;
                    message = $"You are now signed-in as {user.UserId}.";
                }
            }
        }
        catch (Exception ex)
        {
            message = $"Authentication Failed: {ex.Message}";
        }

        // 成功または失敗を表示します。
        await new MessageDialog(message, "Sign-in result").ShowAsync();

        return success;
    }

    public MainPage()
    {
        this.InitializeComponent();

        LoadApplication(new <your_Portable_Class_Library_namespace>.App());
    }
    }
```

Twitter 以外の ID プロバイダーを使用している場合、MobileServiceAuthenticationProvider には別の値を選択してください。

7. LoadApplication() の呼び出しの前にある MainPage クラスのコンストラクター内に次のコード行を追加します。

更新前

MainPage.xaml.cs

```
public MainPage()
{
    this.InitializeComponent();

    LoadApplication(new XamarinHands_on.App());
}
```

更新後

MainPage.xaml.cs

```
public MainPage()
{
    this.InitializeComponent();

    // アプリがロードされる前に、authenticator を初期化します。
    <your_Portable_Class_Library_namespace>.App.Init(this);

    LoadApplication(new <your_Portable_Class_Library_namespace>.App());
}
```

<your_Portable_Class_Library_namespace> を、ポータブル クラス ライブラリの名前空間に置き換えます。

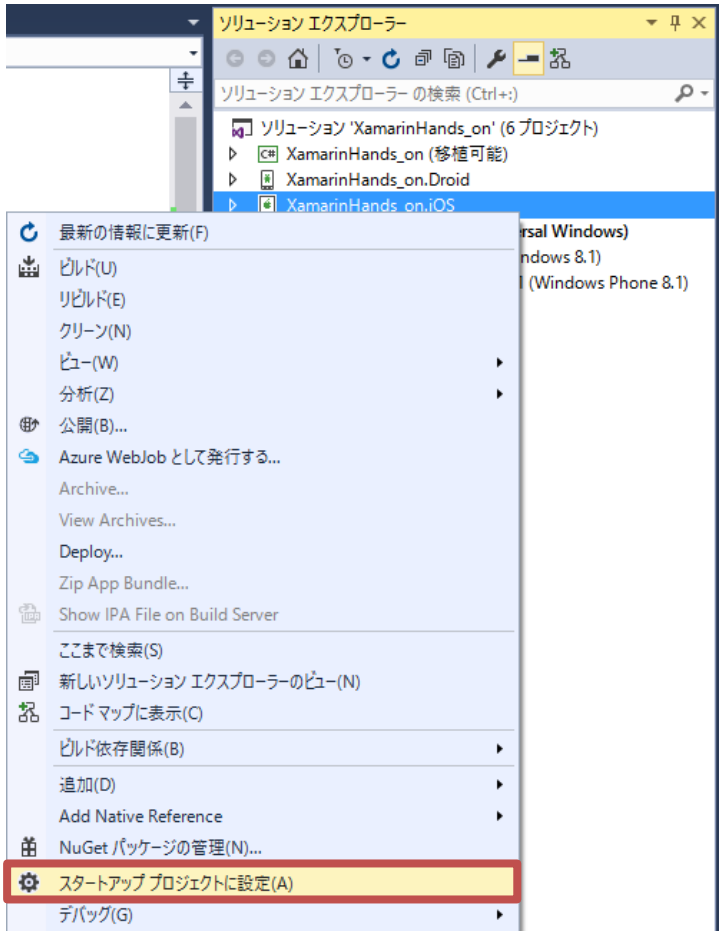
Ctrl+S キーを押して、保存をしてください。

8. アプリをリビルドして実行します。その後で、選択した認証プロバイダーを使用してサインインし、認証されたユーザーとしてデータにアクセスできることを確認します。

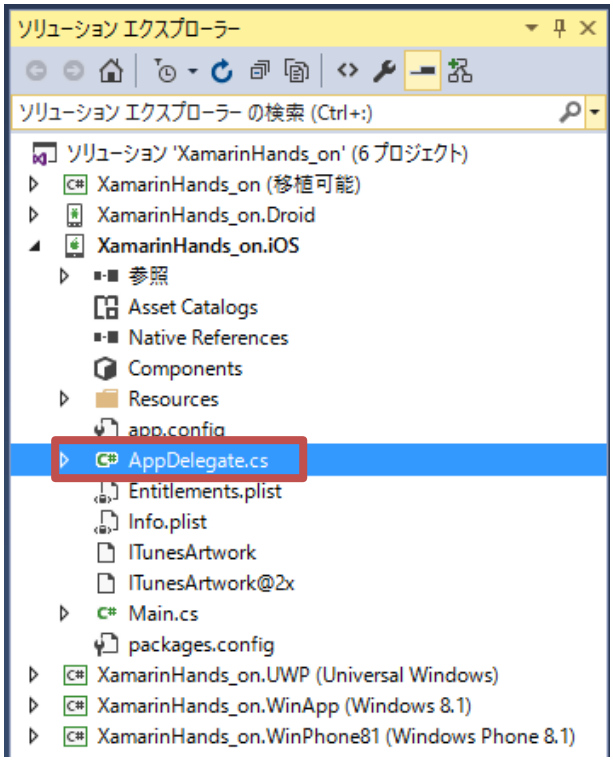
(オプション) iOS アプリに認証を追加する

このセクションでは、iOS アプリ プロジェクト内に IAuthenticate インターフェイスを実装する方法について説明します。iOS のビルド環境が無い場合は、このセクションはスキップしてください。

1. Visual Studio で、iOS プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。



2. F5 キーを押してデバッガーでプロジェクトを開始し、アプリの開始後に、状態コード 401 のハンドルされない例外 (許可されていません) が発生することを確認します。これは、バックエンドへのアクセスが承認済みのユーザーのみに制限されているために発生します。
3. iOS プロジェクトの AppDelegate.cs を開きます。



4. 次の using ステートメントを追加します。

更新前

AppDelegate.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

using Foundation;
using UIKit;
```

更新後

AppDelegate.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

using System.Threading.Tasks;
using Microsoft.WindowsAzure.MobileServices;

using Foundation;
using UIKit;
```

5. 次のように、AppDelegate クラスを更新して IAuthenticate インターフェイスを実装します。

更新前

AppDelegate.cs

```
public partial class AppDelegate :
global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
```

更新後

AppDelegate.cs

```
public partial class AppDelegate :
global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate, IAuthenticate
```

6. 次のように、MobileServiceUser フィールドのほか、IAuthenticate インターフェイスに必要な AuthenticateAsync メソッドを追加して、AppDelegate クラスを更新します。

更新前

AppDelegate.cs

```
public partial class AppDelegate :
global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
{
    public override bool FinishedLaunching (UIApplication app, NSDictionary
options)
    {
```

更新後

AppDelegate.cs

```
public partial class AppDelegate :
global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
{
    // 認証されたユーザーの定義
    private MobileServiceUser user;

    public async Task<bool> AuthenticateAsync()
    {
        var success = false;
        var message = string.Empty;
        try
        {
            // Twitterで サインインを行います。
            if (user == null)
            {
                user = await TodoItemManager.DefaultManager.CurrentClient
```

```
                    .LoginAsync(UIApplication.SharedApplication.KeyWindow.Root
ViewController,
                    MobileServiceAuthenticationProvider.Twitter);
                if (user != null)
                {
                    message = $"You are now signed-in as {user.UserId}.";
                    success = true;
                }
            }
        }
        catch (Exception ex)
        {
            message = ex.Message;
        }

        // 成功または失敗を表示します。
        UIAlertView avAlert = new UIAlertView("Sign-in result", message,
(IUIAlertViewDelegate)null, "OK", null);
        avAlert.Show();

        return success;
    }

    public override bool FinishedLaunching (UIApplication app, NSDictionary
options)
    {
```

Twitter 以外の ID プロバイダーを使用している場合、MobileServiceAuthenticationProvider には別の値を選択してください。

7. LoadApplication() の呼び出しの前の FinishedLaunching メソッドに、次のコード行を追加します。

更新前

AppDelegate.cs

```
public override bool FinishedLaunching (UIApplication app, NSDictionary
options)
{
    // Initialize Azure Mobile Apps
    Microsoft.WindowsAzure.MobileServices.CurrentPlatform.Init();

    // Initialize Xamarin Forms
    global::Xamarin.Forms.Forms.Init ();

    LoadApplication (new App ());

    return base.FinishedLaunching (app, options);
}
```

更新後

AppDelegate.cs

```
public override bool FinishedLaunching (UIApplication app, NSDictionary
options)
{
    // Initialize Azure Mobile Apps
    Microsoft.WindowsAzure.MobileServices.CurrentPlatform.Init();

    // Initialize Xamarin Forms
    global::Xamarin.Forms.Forms.Init ();

    App.Init(this);

    LoadApplication (new App ());

    return base.FinishedLaunching (app, options);
}
```

これにより、アプリの読み込み前に Authenticator が初期化されるようになります。

Ctrl+S キーを押して、保存をしてください。

8. アプリをリビルドして実行します。その後で、選択した認証プロバイダーを使用してサインインし、認証されたユーザーとしてデータにアクセスできることを確認します。

演習 3 : Xamarin.Forms アプリへのプッシュ通知を追加する

この演習では、Azure サービスを使用して、各種ネイティブ デバイス プラットフォーム(Android、iOS、Windows) で実行されている Xamarin.Forms アプリにプッシュ通知を送信する方法について説明します。プッシュ通知は、Azure Notification Hubs を使用して Azure Mobile Apps バックエンドから送信されます。さまざまなプッシュ通知サービス (PNS) を使用して、どのプラットフォームで実行されているデバイスにも同じメッセージを送信できるように、テンプレート登録が使用されます。クロスプラットフォーム プッシュ通知の送信の詳細については、Azure Notification Hubs のドキュメントを参照してください。

対象の Xamarin.Forms アプリでサポートされているすべてのプロジェクトにプッシュ通知を追加します。レコードがバックエンドに挿入されるたびに、プッシュ通知が送信されます。

前提条件

演習 1 を完了している必要があります。

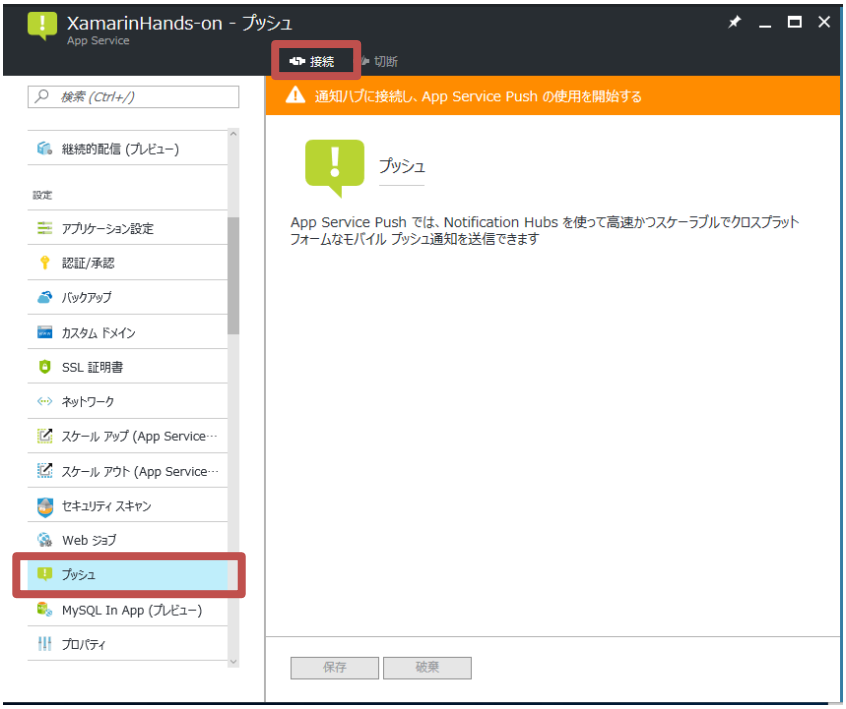
プッシュ通知を iOS デバイスに送信するには、Apple Developer Program メンバーシップが必要です。また、iOS シミュレーターはプッシュ通知をサポートしないため、物理的な iOS デバイスを使用する必要があります。

> **注意**
> Android Emulator for Visual Studio は Google APIs が含まれていないため、プッシュ通知をサポートしません。Android OS が動作する実機をご用意ください。エミュレーターで利用する場合は、Google が配布している Android Virtual Device (AVD) を利用し、Google API を含めた仮想デバイスを用意する必要があります。

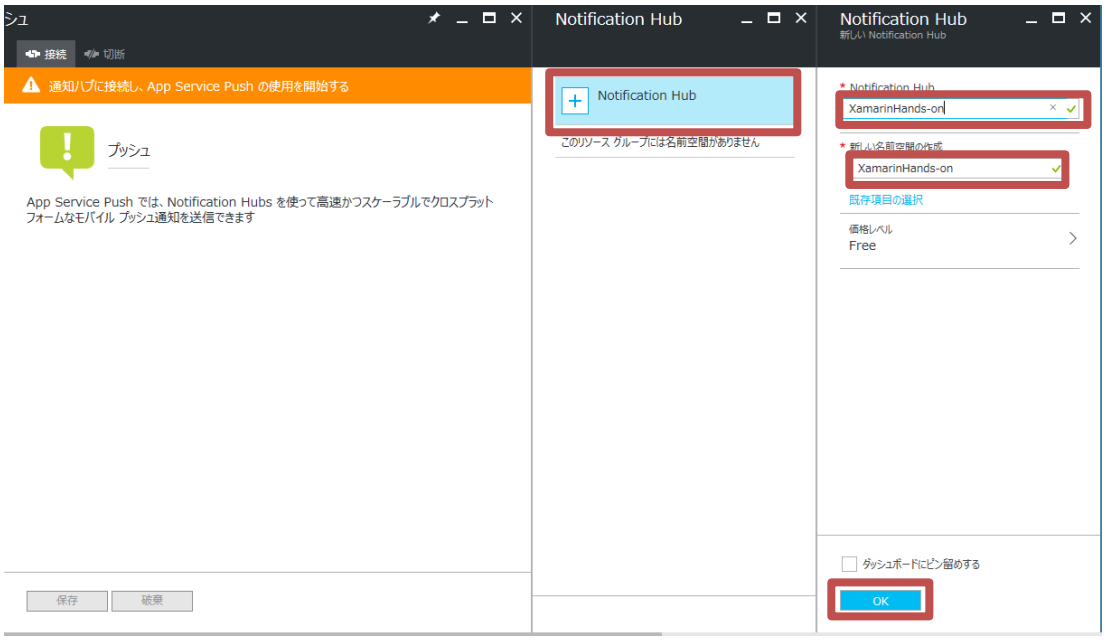
通知ハブを作成する

次の手順に従って、プッシュ通知に使用する新しい通知ハブを作成します。通知ハブが既にある場合は、それをモバイル アプリのバックエンドに接続することもできます。

1. Azure ポータル にログオンし、アプリケーションに移動します。次に、[設定] の [プッシュ] をクリックします。続いて、[接続] をクリックします。



2. さらに [＋Notification Hub] をクリックします。Notification Hub ブレードで、新しい通知ハブの名前を入力します。この名前は、モバイル アプリのバックエンドと同じものにすることができます。既存の名前空間を使用するか、[または新規作成] をクリックして、新しい名前空間の名前を入力してから [OK] をクリックし、デプロイが終わるまで待ちます。



3. デプロイが完了すると、下記のような画面に変わります。

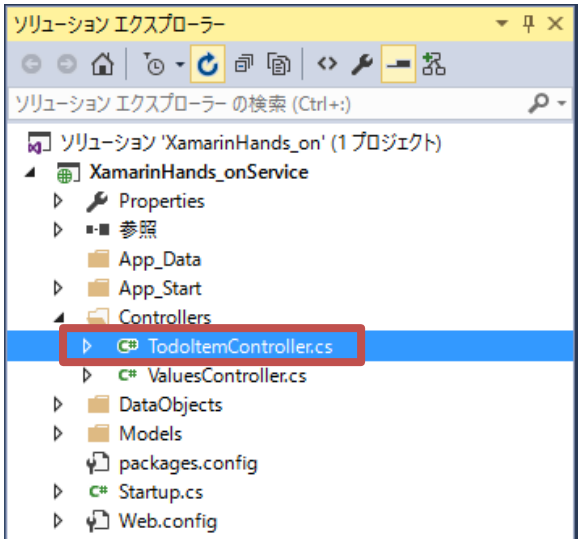
これにより、新しい通知ハブが作成され、それがモバイル アプリに接続されます。通知ハブが既にある場合は、新しい通知ハブを作成せずに、その既存の通知ハブをモバイル アプリに接続してもかまいません。

これで、通知ハブがモバイル アプリのバックエンドに接続されました。後でこの通知ハブを構成して、ネイティブ デバイスにプッシュ通知を送信するプラットフォーム通知サービス (PNS) に接続します。

プッシュ通知を送信するようにサーバー プロジェクトを更新する

このセクションでは、モバイル アプリの既存のバックエンド プロジェクトのコードを更新して、新しい項目が追加されるたびにプッシュ通知を送信するようにします。テンプレート登録の使用によりクライアントはプッシュ通知に登録されるため、単一のプッシュ通知メッセージが、すべてのクライアント プラットフォームに送信されます。各クライアントのテンプレート登録には、*messageParam* パラメーターが含まれます。通知が送信されると、*messageParam* には挿入された項目のテキストを表す文字列が含まれます。Notification Hubs を使用するテンプレートの使用方法の詳細については、「テンプレート」を参照してください。

1. サーバー プロジェクトで、[Controllers > TodoItemController.cs] を開きます。



2. 次の using ステートメントを追加します。

更新前

TodoItemController.cs

```
using System.Linq;
using System.Threading.Tasks;
using System.Web.Http;
using System.Web.Http.Controllers;
using System.Web.Http.OData;
using Microsoft.Azure.Mobile.Server;
using XamarinHands_onService.DataObjects;
using XamarinHands_onService.Models;
```

更新後

TodoItemController.cs

```
using System.Linq;
using System.Threading.Tasks;
using System.Web.Http;
using System.Web.Http.Controllers;
using System.Web.Http.OData;
using Microsoft.Azure.Mobile.Server;
using XamarinHands_onService.DataObjects;
using XamarinHands_onService.Models;

using System.Collections.Generic;
using Microsoft.Azure.NotificationHubs;
using Microsoft.Azure.Mobile.Server.Config;
```

3. 続いて、TodoItemController クラスの PostTodoItem メソッドを、下記のように追記します。これにより、新しい項目が挿入された場合には item.Text を含むテンプレート通知が送信されます。

更新前

TodoItemController.cs

```
public async Task<IHttpActionResult> PostTodoItem(TodoItem item)
{
    TodoItem current = await InsertAsync(item);
    return CreatedAtRoute("Tables", new { id = current.Id }, current);
}
```

更新後

TodoItemController.cs

```
public async Task<IHttpActionResult> PostTodoItem(TodoItem item)
{
    TodoItem current = await InsertAsync(item);
    // サーバー プロジェクト の設定を取得します。
    HttpConfiguration config = this.Configuration;
    MobileAppSettingsDictionary settings =

this.Configuration.GetMobileAppSettingsProvider().GetMobileAppSettings();

    // Notification Hubs 資格情報を取得します。
    string notificationHubName = settings.NotificationHubName;
    string notificationHubConnection = settings
        .Connections[MobileAppSettingsKeys.NotificationHubConnectionString].Co
nnectionString;

    //新しい Notification Hub クライアントを生成します。
    NotificationHubClient hub = NotificationHubClient
    .CreateClientFromConnectionString(notificationHubConnection,
notificationHubName);

    // 通知が送信されると、messageParam には挿入された項目のテキストを表す文字列が
含まれます。
    Dictionary<string, string> templateParams = new Dictionary<string,
string>();
    templateParams["messageParam"] = item.Text + " was added to the list.";

    try
    {
        // プッシュ通知を送信し、結果をログに出力します。
        var result = await hub.SendTemplateNotificationAsync(templateParams);

        // 成功したことをログに出力します。
        config.Services.GetTraceWriter().Info(result.State.ToString());
    }
```

```
    catch (System.Exception ex)
    {
        // 失敗したことをログに出力します。
        config.Services.GetTraceWriter()
            .Error(ex.Message, null, "Push.SendAsync Error");
    }
    return CreatedAtRoute("Tables", new { id = current.Id }, current);
}
```

4. サーバー プロジェクトを発行します。

(オプション) Android プロジェクトを構成して実行する

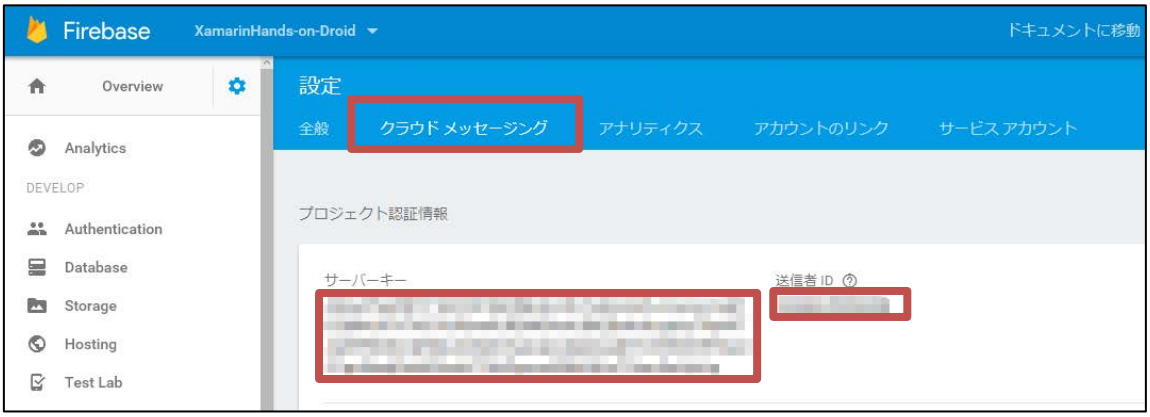
このセクションを完了すると、Android 用の Xamarin.Forms Droid プロジェクトのプッシュ通知を有効にすることができます。

Google Cloud Messaging (GCM) を有効にする

1. Fitbase Console に移動し、Google アカウントの資格情報でサインインし、[プロジェクトを追加] をクリックします。

2. 画面左上の歯車をクリックし、続いて [プロジェクトの設定] をクリックします。

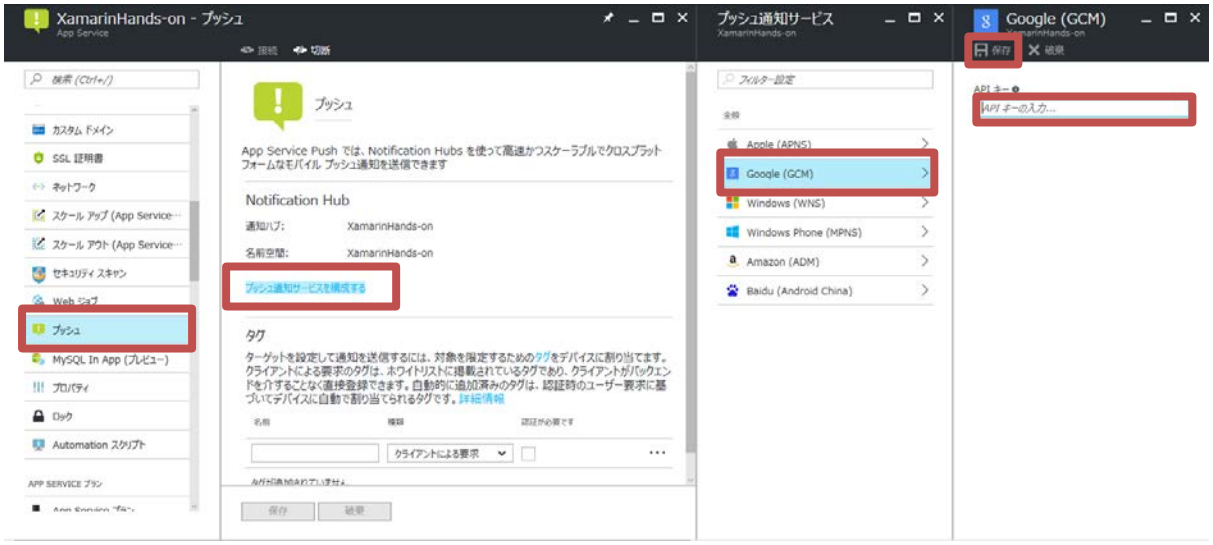
3. [設定] ページの [クラウド メッセージング] タブをクリックし、[プロジェクト認証情報] の [サーバーキー] と [送信者 ID] をメモしておきます。



この API キー値を使用して、Azure が GCM で認証し、アプリの代わりにプッシュ通知を送信できるようにします。

GCM を使用してプッシュ要求を送信するようにモバイル アプリ バックエンドを構成する

1. Azure ポータル にログオンし、アプリケーションに移動します。次に、[設定] の [プッシュ] をクリックします。続いて、[プッシュ通知サービスを構成する] をクリックします。[プッシュ通知サービス] ブレードが表示されるので、[Google (GCM)] をクリックし、[API キー]に先ほどの **サーバーキー** を入力し、最後に [保存] をクリックします。

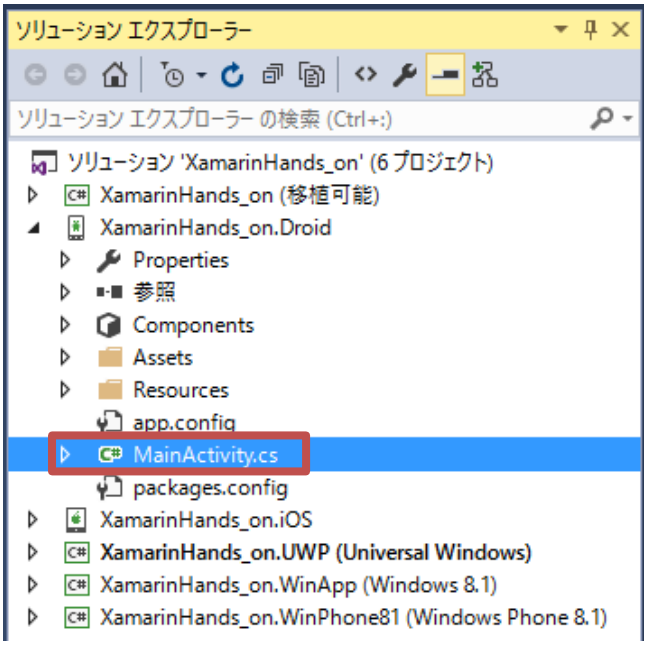


これで、モバイル アプリ バックエンドは、通知ハブを使用している Android デバイスで実行中のアプリに GCM を使用してプッシュ通知を使用するように構成されました。

Android プロジェクトにプッシュ通知を追加する

Google Cloud Messaging (GCM) を使用するようにバックエンドが構成されている場合は、GCM へのアプリの登録、モバイル アプリ バックエンドを介した Azure Notification Hubs によるプッシュ通知へのアプリの登録、アプリによる通知の受信を可能にするコンポーネントとコードをクライアントに追加できます。

1. Droid プロジェクトで [コンポーネント] フォルダーを右クリックし、[コンポーネントをさらに取得する...] をクリックします。Google Cloud Messaging Client コンポーネントを検索し、それをプロジェクトに追加します。このコンポーネントは、Xamarin Android プロジェクトのプッシュ通知をサポートします。
2. MainActivity.cs プロジェクト ファイルを開きます。



3. 次の using ステートメントをファイルの先頭に追加します。

更新前

MainActivity.cs

```
using System;
using System.Threading.Tasks;
using Microsoft.WindowsAzure.MobileServices;

using Android.App;
using Android.Content;
using Android.Content.PM;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
```

更新後

MainActivity.cs

```
using System;
using System.Threading.Tasks;
using Microsoft.WindowsAzure.MobileServices;

using Android.App;
using Android.Content;
using Android.Content.PM;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;

using Gcm.Client;
```

4. MainActivity クラスの LoadApplication の呼び出し後、次のコードを OnCreate メソッドに追加します。また、新しい CreateAndShowDialog ヘルパーメソッドを追加します。

更新前

MainActivity.cs

```
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);

    // Initialize Azure Mobile Apps
    Microsoft.WindowsAzure.MobileServices.CurrentPlatform.Init();

    // Initialize Xamarin Forms
```

```
    global::Xamarin.Forms.Forms.Init (this, bundle);

    // アプリがロードされる前に、authenticator を初期化します。
    App.Init((IAuthenticate)this);

    // Load the main application
    LoadApplication (new App ());
}
```

更新後

MainActivity.cs

```
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);

    // Initialize Azure Mobile Apps
    Microsoft.WindowsAzure.MobileServices.CurrentPlatform.Init();

    // Initialize Xamarin Forms
    global::Xamarin.Forms.Forms.Init (this, bundle);

    // アプリがロードされる前に、authenticator を初期化します。
    App.Init((IAuthenticate)this);

    // Load the main application
    LoadApplication (new App ());

    try
    {
        // Check to ensure everything's setup right
        GcmClient.CheckDevice(this);
        GcmClient.CheckManifest(this);

        // Register for push notifications
        System.Diagnostics.Debug.WriteLine("Registering...");
        GcmClient.Register(this, PushHandlerBroadcastReceiver.SENDER_IDS);
    }
    catch (Java.Net.MalformedURLException)
    {
        CreateAndShowDialog("There was an error creating the client. Verify
the URL.", "Error");
    }
    catch (Exception e)
    {
        CreateAndShowDialog(e.Message, "Error");
    }
}

private void CreateAndShowDialog(String message, String title)
```

```
{
    AlertDialog.Builder builder = new AlertDialog.Builder(this);

    builder.SetMessage (message);
    builder.SetTitle (title);
    builder.Create().Show ();
}
```

5. MainActivity クラスに次のコードを追加します。

更新前

MainActivity.cs

```
public class MainActivity :
global::Xamarin.Forms.Platform.Android.FormsApplicationActivity
{
```

更新後

MainActivity.cs

```
public class MainActivity :
global::Xamarin.Forms.Platform.Android.FormsApplicationActivity
{
    // Create a new instance field for this activity.
    static MainActivity instance = null;

    // Return the current activity instance.
    public static MainActivity CurrentActivity { get { return instance; } }
```

これにより、現在の MainActivity インスタンスが公開され、メイン UI スレッドで実行できるようになります。

6. OnCreate メソッドの先頭で、変数 instance を次のように初期化します。

更新前

MainActivity.cs

```
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);
```

更新後

MainActivity.cs

```
protected override void OnCreate (Bundle bundle)
{
    // Set the current instance of MainActivity.
    instance = this;
```

```
    base.OnCreate (bundle);
```

7. GcmService.cs という名前の Droid プロジェクトに新しいクラス ファイルを追加します。ファイルの先頭にある using ステートメントを、下記の内容で入れ替えます。

更新前

GcmService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Android.App;
using Android.Content;
using Android.OS;
using Android.Runtime;
using Android.Views;
using Android.Widget;
```

更新後

GcmService.cs

```
using Android.App;
using Android.Content;
using Android.Media;
using Android.Support.V4.App;
using Android.Util;
using Gcm.Client;
using Microsoft.WindowsAzure.MobileServices;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
```

8. ファイルの先頭の using ステートメントと namespace 宣言の間に、次のアクセス許可要求を追加します。

更新前

GcmService.cs

```
using System.Text;

namespace xxxxxxx.Droid
{
```

更新後

GcmService.cs

```
using System.Text;

[assembly: Permission(Name = "@PACKAGE_NAME@.permission.C2D_MESSAGE")]
[assembly: UsesPermission(Name = "@PACKAGE_NAME@.permission.C2D_MESSAGE")]
[assembly: UsesPermission(Name =
"com.google.android.c2dm.permission.RECEIVE")]
[assembly: UsesPermission(Name = "android.permission.INTERNET")]
[assembly: UsesPermission(Name = "android.permission.WAKE_LOCK")]
//GET_ACCOUNTS is only needed for android versions 4.0.3 and below
[assembly: UsesPermission(Name = "android.permission.GET_ACCOUNTS")]

namespace xxxxxxx.Droid
{
```

9. 名前空間の中に次のクラス定義を追加します。

更新前

GcmService.cs

```
namespace xxxxxxx.Droid
{
    class GcmService
    {
    }
}
```

更新後

GcmService.cs

```
namespace xxxxxxx.Droid
{
    [BroadcastReceiver(Permission =
Gcm.Client.Constants.PERMISSION_GCM_INTENTS)]
    [IntentFilter(new string[]
{ Gcm.Client.Constants.INTENT_FROM_GCM_MESSAGE }, Categories = new string[]
{ "@PACKAGE_NAME@" })]
    [IntentFilter(new string[]
{ Gcm.Client.Constants.INTENT_FROM_GCM_REGISTRATION_CALLBACK }, Categories =
new string[] { "@PACKAGE_NAME@" })]
    [IntentFilter(new string[]
{ Gcm.Client.Constants.INTENT_FROM_GCM_LIBRARY_RETRY }, Categories = new
string[] { "@PACKAGE_NAME@" })]
    public class PushHandlerBroadcastReceiver :
GcmBroadcastReceiverBase<GcmService>
    {
```

```
        public static string[] SENDER_IDS = new string[]
{ "<PROJECT_NUMBER>" };
    }

    class GcmService
    {
    }

}
```

<PROJECT_NUMBER> を、先ほどメモした 送信者 ID に置き換えます。

10. 空の GcmService クラスを、新しいブロードキャスト レシーバーを使用する次のコードに置き換えます。

更新前

GcmService.cs

```
class GcmService
{
}
```

更新後

GcmService.cs

```
[Service]
public class GcmService : GcmServiceBase
{
    public static string RegistrationID { get; private set; }

    public GcmService()
        : base(PushHandlerBroadcastReceiver.SENDER_IDS) { }
}
```

11. GcmService クラスの、OnRegistered イベント ハンドラーをオーバーライドし、Register メソッドを実装する次のコードを GcmService クラスに追加します。

更新前

GcmService.cs

```
[Service]
public class GcmService : GcmServiceBase
{
    public static string RegistrationID { get; private set; }

    public GcmService()
        : base(PushHandlerBroadcastReceiver.SENDER_IDS) { }
}
```

更新後

GcmService.cs

```
[Service]
public class GcmService : GcmServiceBase
{
    public static string RegistrationID { get; private set; }

    public GcmService()
        : base(PushHandlerBroadcastReceiver.SENDER_IDS) { }

    protected override void OnRegistered(Context context, string
registrationId)
    {
        Log.Verbose("PushHandlerBroadcastReceiver", "GCM Registered: " +
registrationId);
        RegistrationID = registrationId;

        var push = TodoItemManager.DefaultManager.CurrentClient.GetPush();

        MainActivity.CurrentActivity.RunOnUiThread(() => Register(push,
null));
    }

    public async void Register(Push push, IEnumerable<string> tags)
    {
        try
        {
            const string templateBodyGCM =
                    "{\"data\":{\"message\":\"$(messageParam)\"}}";

            JObject templates = new JObject();
            templates["genericMessage"] = new JObject
            {
                {"body", templateBodyGCM}
            };

            await push.RegisterAsync(RegistrationID, templates);
            Log.Info("Push Installation Id", push.InstallationId.ToString());
        }
        catch (Exception ex)
        {
            System.Diagnostics.Debug.WriteLine(ex.Message);
            Debugger.Break();
        }
    }
}
```

12. 続いて、OnMessage を実装する次のコードを追加します。

更新前

GcmService.cs

```
[Service]
public class GcmService : GcmServiceBase
{
    // ・・・・省略・・・・
}
```

更新後

GcmService.cs

```
[Service]
public class GcmService : GcmServiceBase
{
    // ・・・・省略・・・・

    protected override void OnMessage(Context context, Intent intent)
    {
        Log.Info("PushHandlerBroadcastReceiver", "GCM Message Received!");

        var msg = new StringBuilder();

        if (intent != null && intent.Extras != null)
        {
            foreach (var key in intent.Extras.KeySet())
                msg.AppendLine(key + "=" + intent.Extras.Get(key).ToString());
        }

        //Store the message
        var prefs = GetSharedPreferences(context.PackageName,
                                         FileCreationMode.Private);
        var edit = prefs.Edit();
        edit.PutString("last_msg", msg.ToString());
        edit.Commit();

        string message = intent.Extras.GetString("message");
        if (!string.IsNullOrEmpty(message))
        {
            createNotification("New todo item!", "Todo item: " + message);
            return;
        }

        string msg2 = intent.Extras.GetString("msg");
        if (!string.IsNullOrEmpty(msg2))
        {
            createNotification("New hub message!", msg2);
            return;
        }

        createNotification("Unknown message details", msg.ToString());
    }
```

```
    void createNotification(string title, string desc)
    {
        //Create notification
        var notificationManager =
            GetSystemService(Context.NotificationService) as
NotificationManager;

        //Create an intent to show ui
        var uiIntent = new Intent(this, typeof(MainActivity));

        //Use Notification Builder
        NotificationCompat.Builder builder = new
NotificationCompat.Builder(this);

        //Create the notification
        //we use the pending intent,
        //passing our ui intent over which will get called
        //when the notification is tapped.
        var notification =
         builder.SetContentIntent(PendingIntent.GetActivity(this, 0, uiIntent,
0))
                .SetSmallIcon(Android.Resource.Drawable.SymActionEmail)
                .SetTicker(title)
                .SetContentTitle(title)
                .SetContentText(desc)

                //Set the notification sound
                .SetSound(RingtoneManager.GetDefaultUri(RingtoneType.Notificat
ion))

                //Auto cancel will remove the notification once
                //the user touches it
                .SetAutoCancel(true).Build();

        //Show the notification
        notificationManager.Notify(1, notification);
    }
}
```

これにより、受信した通知が処理され、表示される通知マネージャーに送信されます。

13. また、GcmServiceBase では、OnUnRegistered および OnError ハンドラー メソッドを実装する必要があります。これらは、次のように実装できます。

更新前

GcmService.cs

```
[Service]
public class GcmService : GcmServiceBase
{
```

```
    // ・・・・省略・・・・
}
```

更新後

GcmService.cs

```
[Service]
public class GcmService : GcmServiceBase
{
    // ・・・・省略・・・・

    protected override void OnUnRegistered(Context context, string
registrationId)
    {
        Log.Error("PushHandlerBroadcastReceiver",
                  $"Unregistered RegisterationId : {registrationId}");
    }

    protected override void OnError(Context context, string errorId)
    {
        Log.Error("PushHandlerBroadcastReceiver", $"GCM Error: {errorId}");
    }
}
```

これで、Android デバイスまたはエミュレーターで実行されているアプリでプッシュ通知をテストする準備が整いました。

Android アプリでプッシュ通知をテストする

最初の 2 つの手順は、エミュレーターでテストする場合にのみ必要です。

次に示すように Android Virtual Device (AVD) Manager で Google API がターゲットとして設定された仮想デバイスに対してデプロイまたはデバッグする必要があります。

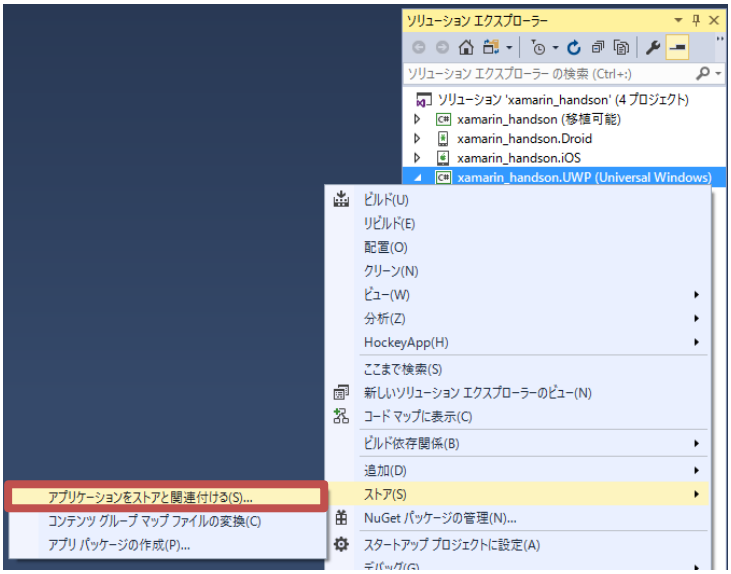
1. [Apps] 、[Settings] 、[Add account] の順にクリックして、Google アカウントを Android デバイスに追加し、プロンプトに従って既存の Google アカウントをデバイスに追加し、新しいアカウントを作成します。
2. Visual Studio で、Droid プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
3. [実行] をクリックしてプロジェクトをビルドし、Android デバイスまたはエミュレーターでアプリを開始します。
4. アプリケーションで、タスクを入力し、プラス (+) アイコンをクリックします。
5. 項目が追加されたときに、通知が受信されていることを確認します。

Windows プロジェクトを構成して実行する

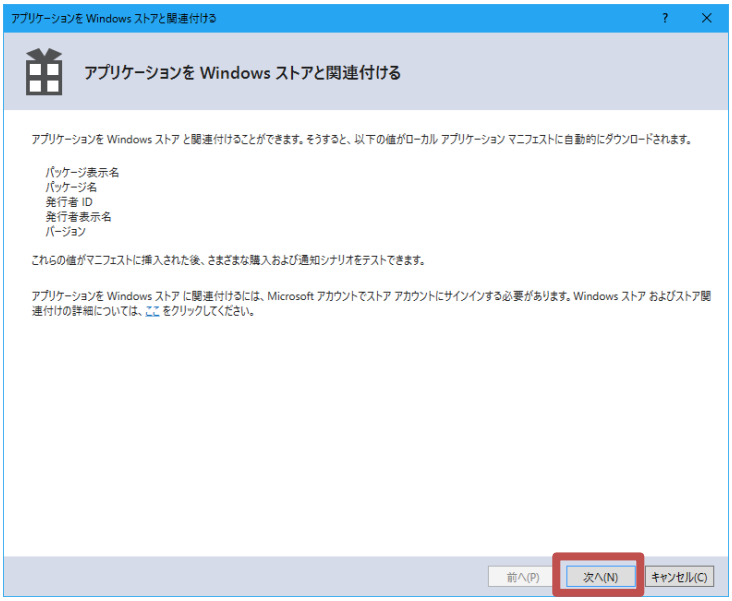
このセクションでは、Windows デバイス用の Xamarin.Forms ユニバーサル Windows プラットフォーム (UWP) プロジェクトを実行します。

WNS を使用して Windows アプリをプッシュ通知に登録する

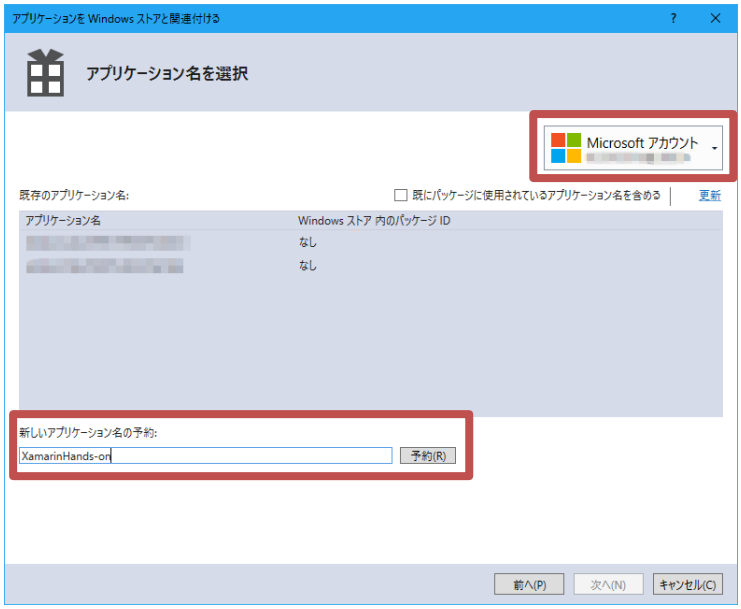
1. Visual Studio の [ソリューション エクスプローラー] で、UWP アプリ プロジェクトを右クリックし、[ストア] 、[アプリケーションをストアと関連付ける] の順にクリックします。



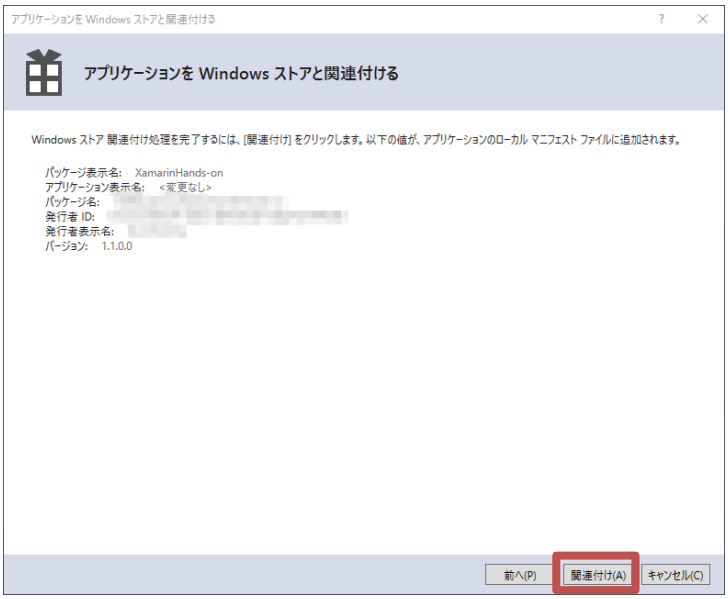
2. ウィザード画面が表示されるので、[次へ] をクリックします。



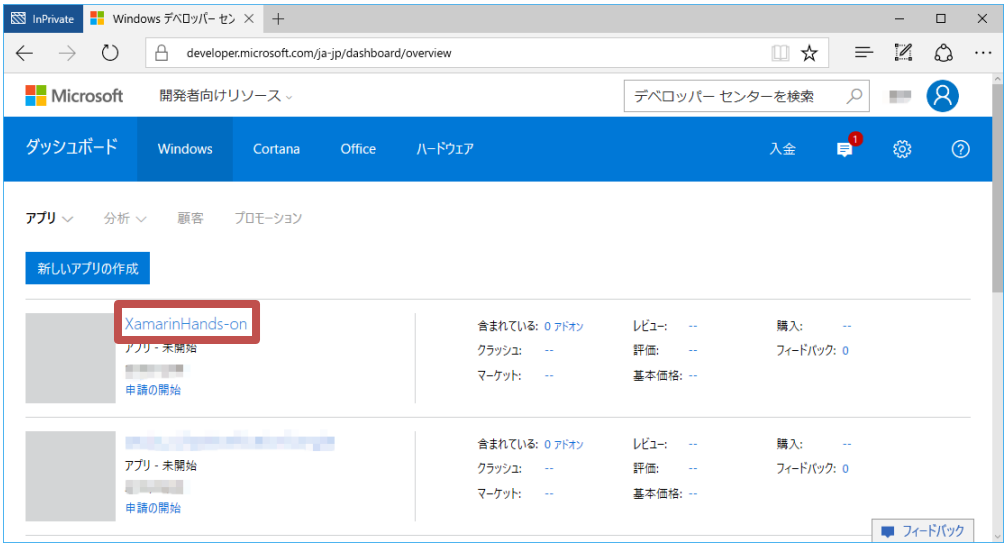
3. [アプリケーション名を選択] 画面で、Microsoft アカウントでサインインし、[新しいアプリケーション名の予約] にアプリの名前を入力し、[予約] をクリックします。



4. アプリの登録が正常に予約されたら、新しいアプリ名を選択し、[次へ] をクリックし、[関連付け] をクリックします。この操作により、必要な Windows ストア登録情報がアプリケーション マニフェストに追加されます。



5. [Windows デベロッパー センター] に移動し、Microsoft アカウントでサインインし、[アプリ] で関連付けをしたアプリをクリックします。



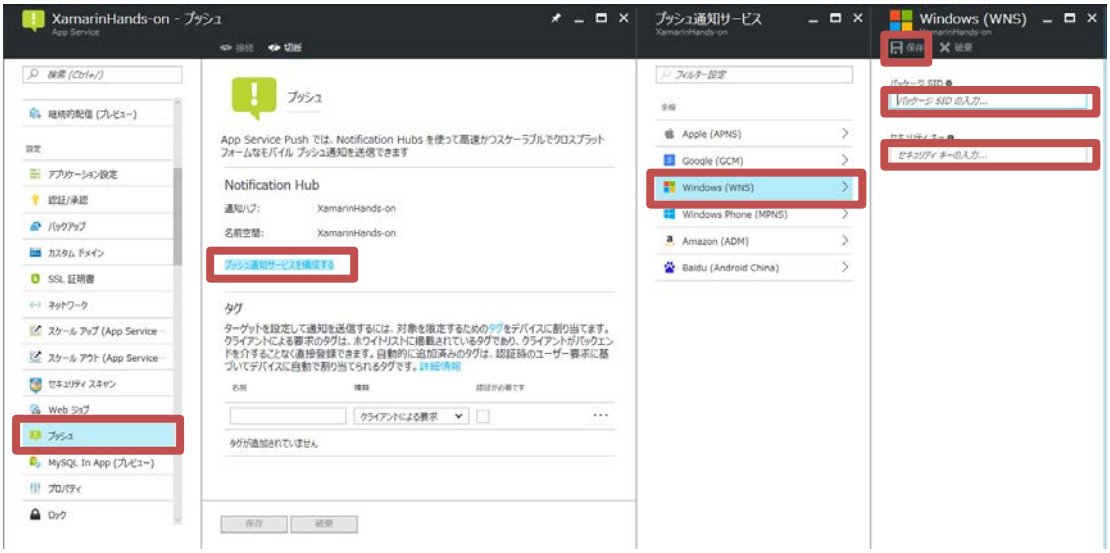
6. 続いて、[サービス] 、[プッシュ通知] のリンクをクリックすると、[プッシュ通知] の下に [WNS/MPNS] のリンクが表示されますので、[WNS/MPNS] をクリックします。

7. [プッシュ通知] ページで、[Windows プッシュ通知サービスと Microsoft Azure Mobile Apps] の下にある [Live サービス サイト] をクリックします。

8. [マイ アプリケーション] に表示されている、[パッケージ SID] 値と [アプリケーション シークレット] の現在値をメモします。

Windows 用に通知ハブを構成する

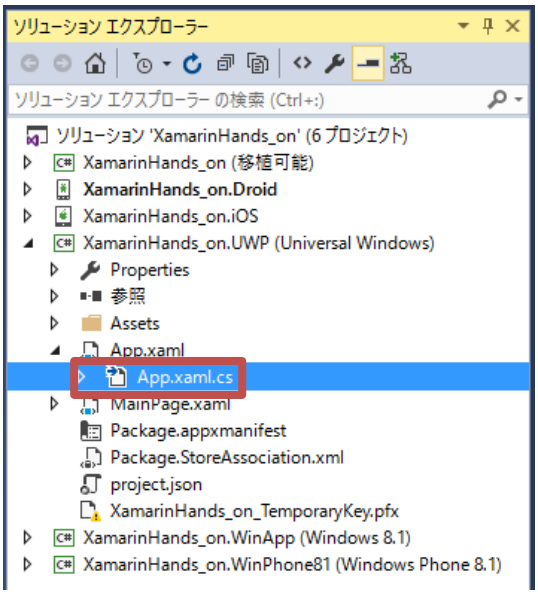
1. Azure ポータル にログオンし、アプリケーションに移動します。次に、[設定] の [プッシュ] をクリックします。続いて、[プッシュ通知サービスを構成する] をクリックします。[プッシュ通知サービス] ブレードが表示されるので、[Windows (WNS)] をクリックし、[パッケージ SID] に先ほどメモした **パッケージ SID** を、[セキュリティー キー]には **アプリケーション シークレット** を入力し、最後に [保存] をクリックします。



バックエンドは WNS を使用してプッシュ通知を送信するよう構成されました。

Windows アプリにプッシュ通知を追加する

1. Visual Studio の UWP プロジェクトで App.xaml.cs を開きます。



2. 次の using ステートメントを追加します。

更新前

App.xaml.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.ApplicationModel;
using Windows.ApplicationModel.Activation;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;
```

更新後

App.xaml.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.ApplicationModel;
using Windows.ApplicationModel.Activation;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

using Newtonsoft.Json.Linq;
using Microsoft.WindowsAzure.MobileServices;
using System.Threading.Tasks;
using Windows.Networking.PushNotifications;
using <your_Portable_Class_Library_namespace>;
```

<your_Portable_Class_Library_namespace> を、ポータブル クラス ライブラリの名前空間に置き換えます。

3. App.xaml.cs で、次の InitNotificationsAsync メソッドを追加します。

更新前

App.xaml.cs

```
sealed partial class App : Application
{
    public App()
    {
        this.InitializeComponent();
        this.Suspending += OnSuspending;
    }
```

更新後

App.xaml.cs

```
sealed partial class App : Application
{
    private async Task InitNotificationsAsync()
    {
        var channel = await PushNotificationChannelManager
            .CreatePushNotificationChannelForApplicationAsync();
        const string templateBodyWNS =
            "<toast><visual><binding template=\"ToastText01\"><text
id=\"1\">$(messageParam)</text></binding></visual></toast>";

        JObject headers = new JObject();
        headers["X-WNS-Type"] = "wns/toast";

        JObject templates = new JObject();
        templates["genericMessage"] = new JObject
        {
            { "body", templateBodyWNS},
            { "headers", headers} // Needed for WNS.
        };

        await TodoItemManager.DefaultManager.CurrentClient.GetPush()
            .RegisterAsync(channel.Uri, templates);
    }

    public App()
    {
        this.InitializeComponent();
        this.Suspending += OnSuspending;
    }
```

このメソッドによって、プッシュ通知チャネルが取得され、対象の通知ハブからテンプレート通知を受け取るためのテンプレートが登録されます。*messageParam* をサポートするテンプレート通知がこのクライアントに配信されるようになります。

4. App.xaml.cs で、async 修飾子を追加して OnLaunched イベント ハンドラー メソッドの定義を更新します。その後で、メソッドの末尾に次のコード行を追加します。

   更新前

   App.xaml.cs

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{

     // ・・・・省略・・・・

    // Ensure the current window is active
    Window.Current.Activate();
}
```

   更新後

   App.xaml.cs

```
protected override async void OnLaunched(LaunchActivatedEventArgs e)
{

     // ・・・・省略・・・・

    // Ensure the current window is active
    Window.Current.Activate();

    await InitNotificationsAsync();
}
```

   これにより、アプリの起動時に毎回プッシュ通知登録が作成または更新されるようになります。これを行うことは、WNS プッシュ チャネルが常にアクティブであることを保証するために重要です。

5. アプリをビルドし、エラーがないことを確認します。これで、クライアント アプリケーションが、モバイル アプリ バックエンドから送信されるテンプレート通知に登録されました。

Windows アプリでプッシュ通知をテストする

1. Visual Studio で、UWP プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. [実行] ボタンを押してプロジェクトをビルドし、アプリケーションを開始します。
3. アプリで新しい todoitem の名前を入力し、プラス (+) アイコンをクリックして追加します。
4. 項目が追加されたときに、通知が受信されていることを確認します。

(オプション) iOS プロジェクトを構成して実行する

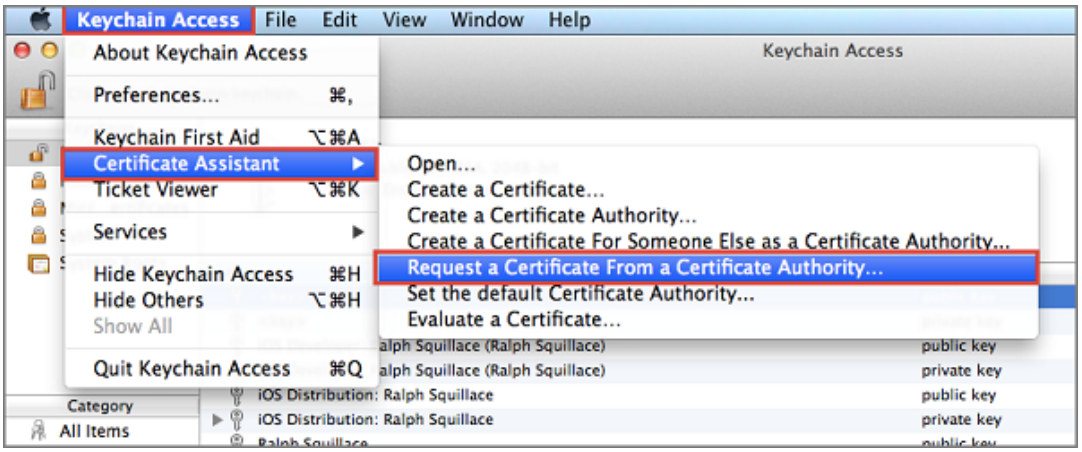
このセクションでは、iOS デバイス用の Xamarin iOS プロジェクトを実行します。iOS デバイスを使用していない場合は、このセクションを省略できます。

Apple Push Notification サービス (APNS) を介してアプリ用にプッシュ通知を登録するには、新しいプッシュ証明書、アプリケーション ID、プロジェクトのプロビジョニング プロファイルを Apple の開発者ポータルで作成する必要があります。アプリケーション ID には、プッシュ通知をアプリケーションで送受信できるようにする構成設定が含まれます。これらの設定には、プッシュ通知を送受信するときに必要な Apple Push Notification サービス (APNS) と認証するためのプッシュ通知証明書が含まれます。これらの概念の詳細については、Apple Push Notification サービスの公式ドキュメントを参照してください。

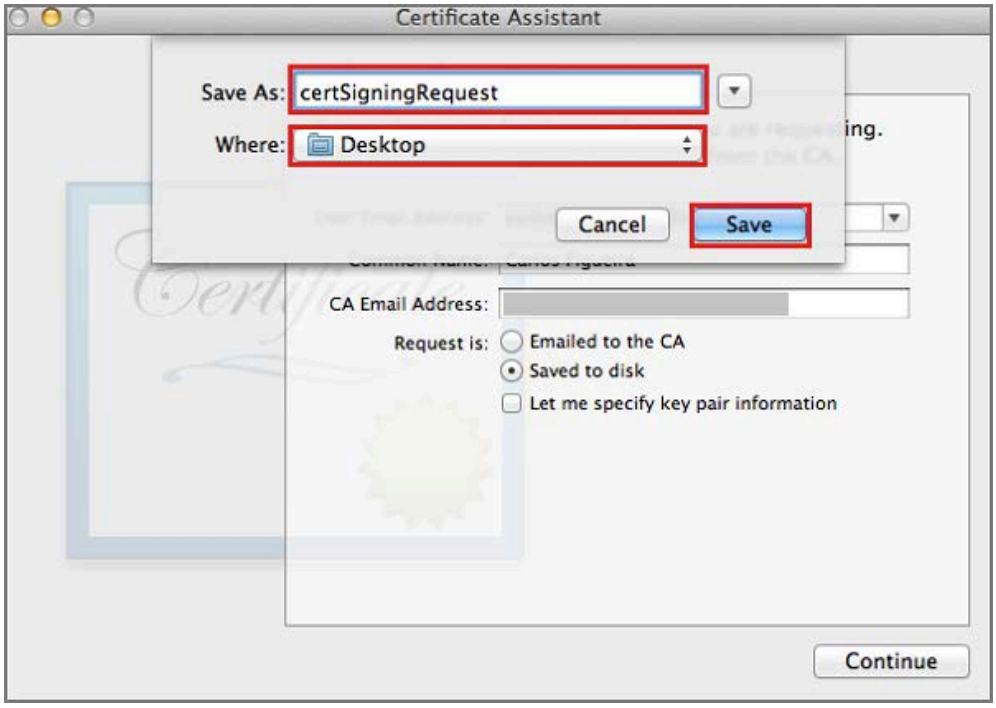
プッシュ証明書用に証明書の署名要求ファイルを生成する

次の手順では、証明書の署名要求の作成手順について説明します。これは、APNS で使用するプッシュ証明書の生成に使用されます。

1. Mac OS で、キーチェーン アクセス ツールを実行します。これは、Launchpad の [ユーティリティー] フォルダーまたは [その他] フォルダーから開くことができます。
2. [キーチェーン アクセス] をクリックし、[証明書アシスタント] を展開して、[認証局に証明書を要求] をクリックします。



3. [ユーザのメールアドレス] と [通称] を選択し、[ディスクに保存] が選択されていることを確認して、[続ける] をクリックします。必要ではないため、"CA のメール アドレス" フィールドは空白のままにします。
4. [名前] に証明書署名要求 (CSR) ファイルの名前を入力し、[場所] で保存場所を選択して [保存] をクリックします。



指定選択した場所に CSR ファイルが保存されます。既定の場所は [デスクトップ] です。このファイル用に選択した場所を忘れないでください。

アプリケーションをプッシュ通知に登録する

Apple でアプリケーション用の新しいアプリケーション ID を作成し、さらにそれをプッシュ通知用に構成します。

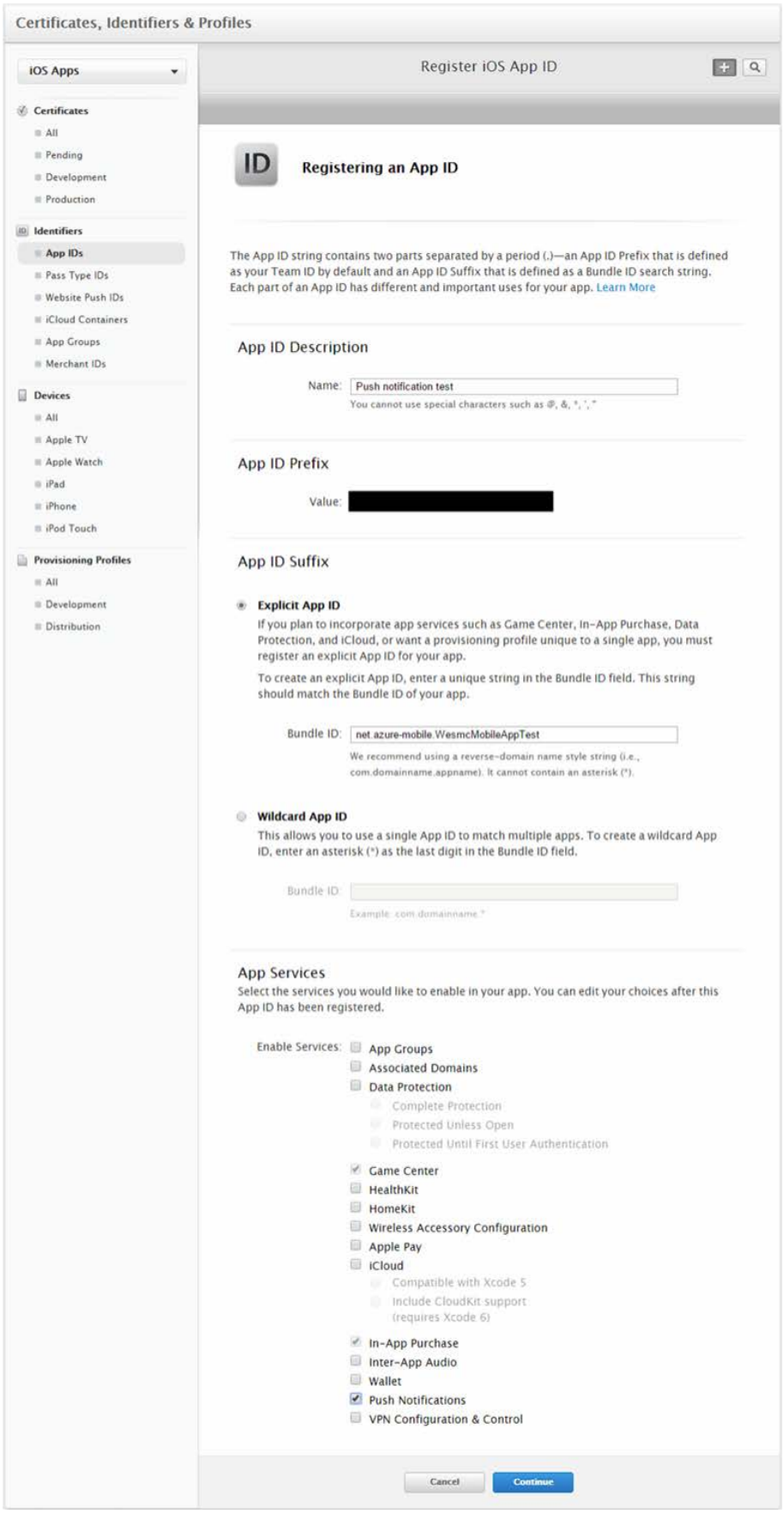
1. Apple Developer センターで iOS Provisioning Portal に移動し、Apple ID でログインして、[Identifiers] をクリックし、[App IDs] をクリックします。最後に、+ 記号をクリックして新しいアプリを登録します。

2. 新しいアプリの次の 3 つのフィールドを更新し、[Continue] をクリックします。

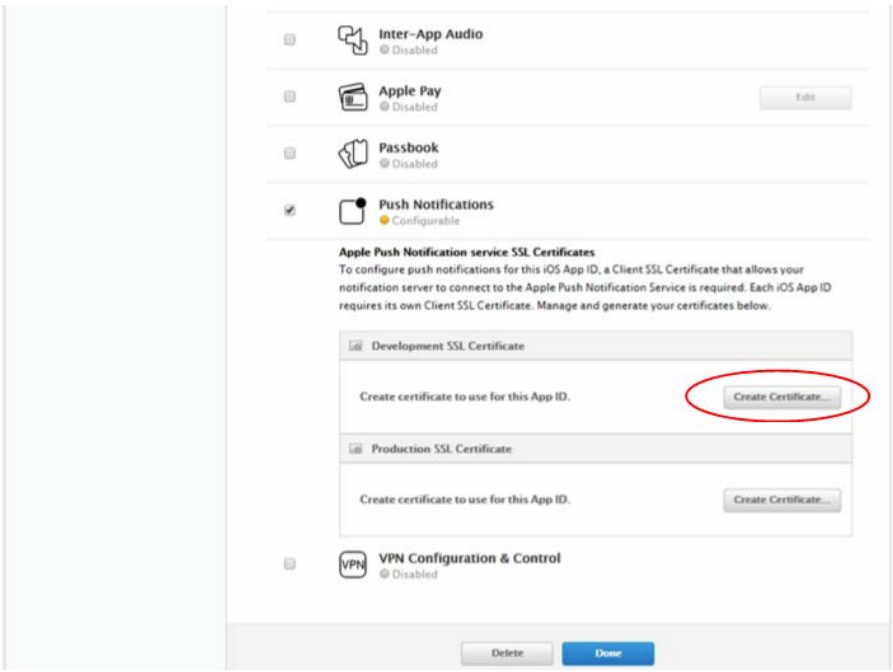
- [Name] : [App ID Description] セクションの [Name] フィールドに、アプリのわかりやすい名前を入力します。

- [Bundle Identifier] : [Explicit App ID] セクションに、アプリ ディストリビューション ガイドで説明したように <Organization Identifier>.<Product Name> の形式でバンドル ID を入力します。これは、アプリ用の Xamarin プロジェクトで使用されているものとも一致する必要があります。

- [Push Notifications] : [App Services] セクションの [Push Notifications] オプションをオンにします。


Certificates, Identifiers & Profiles
iOS Apps
Certificates
All
Pending
Development
Production
Identifiers
App IDs
Pass Type IDs
Website Push IDs
iCloud Containers
App Groups
Merchant IDs
Devices
All
Apple TV
Apple Watch
iPad
iPhone
iPod Touch
Provisioning Profiles
All
Development
Distribution
Register iOS App ID
Registering an App ID
The App ID string contains two parts separated by a period (.)—an App ID Prefix that is defined as your Team ID by default and an App ID Suffix that is defined as a Bundle ID search string. Each part of an App ID has different and important uses for your app. Learn More
App ID Description
Name: Push notification test
You cannot use special characters such as @, &, *, ', "
App ID Prefix
Value:
App ID Suffix
Explicit App ID
If you plan to incorporate app services such as Game Center, In-App Purchase, Data Protection, and iCloud, or want a provisioning profile unique to a single app, you must register an explicit App ID for your app.
To create an explicit App ID, enter a unique string in the Bundle ID field. This string should match the Bundle ID of your app.
Bundle ID: net.azure-mobile.WesmcMobileAppTest
We recommend using a reverse-domain name style string (i.e., com.domainname.appname). It cannot contain an asterisk (*).
Wildcard App ID
This allows you to use a single App ID to match multiple apps. To create a wildcard App ID, enter an asterisk (*) as the last digit in the Bundle ID field.
Bundle ID:
Example: com.domainname.*
App Services
Select the services you would like to enable in your app. You can edit your choices after this App ID has been registered.
Enable Services:
App Groups
Associated Domains
Data Protection
Complete Protection
Protected Unless Open
Protected Until First User Authentication
Game Center
HealthKit
HomeKit
Wireless Accessory Configuration
Apple Pay
iCloud
Compatible with Xcode 5
Include CloudKit support (requires Xcode 6)
In-App Purchase
Inter-App Audio
Wallet
Push Notifications
VPN Configuration & Control
Cancel
Continue

3. [Confirm your App ID] 画面で設定を確認し、確認後 [Submit] をクリックします。
4. 新しいアプリケーション ID を送信すると、[Registration complete] 画面が表示されます。[Done] をクリックします。
5. Developer センターで、アプリケーション ID の一覧から作成したアプリケーション ID を見つけ、その行をクリックします。アプリケーション ID 行をクリックすると、アプリケーションの詳細が表示されます。画面の下部にある [Edit] をクリックします。
6. 画面の下部までスクロールし、[Development Push SSL Certificate] セクションの [Create Certificate] ボタンをクリックします。



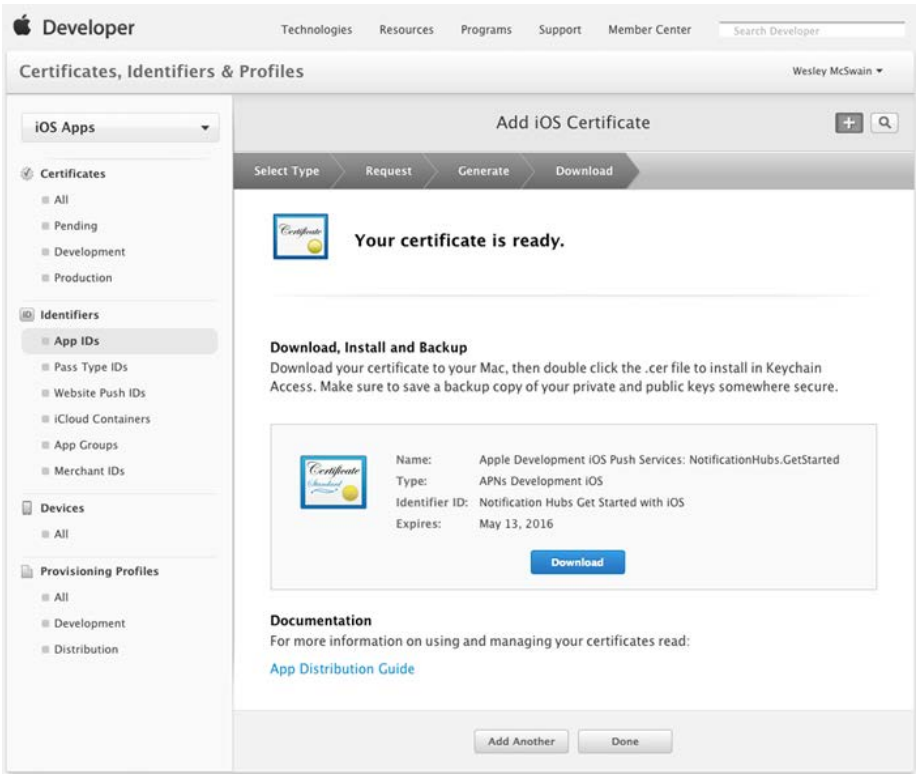
これで、"Add iOS Certificate" アシスタントが表示されます。

メモ:

このチュートリアルでは開発証明書を使用します。運用証明書の場合も同じ処理を行います。通知の送信と同じ証明書の種類を使用するようにします。

7. [Choose File] をクリックして、プッシュ証明書用に CSR を保存した場所に移動します。次いで、[Generate] をクリックします。

8. ポータルで証明書が作成されたら [Download] ボタンをクリックします。

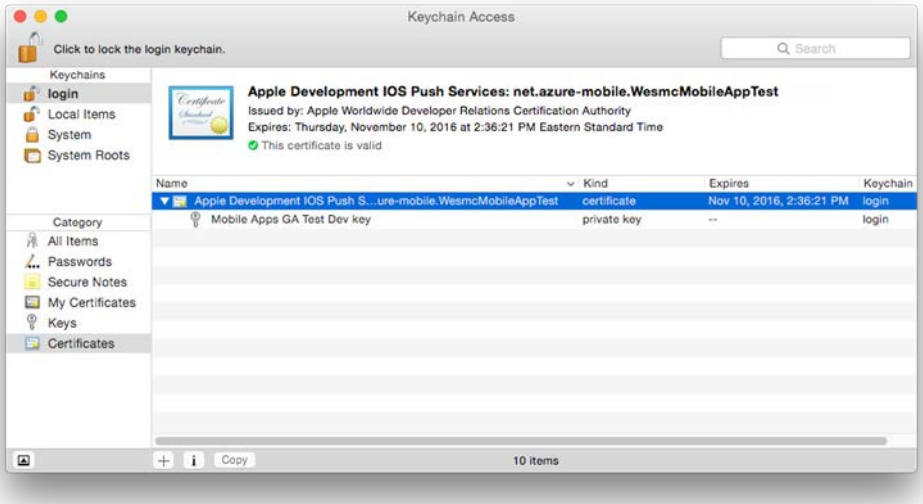


ここまで、署名証明書がダウンロードされ、コンピューターの Downloads フォルダーに保存されます。

メモ:

既定では、ダウンロードした開発証明書ファイルの名前は **aps_development.cer** になっています。

9. ダウンロードしたプッシュ証明書 **aps_development.cer** をダブルクリックします。下図のように、新しい証明書が キーチェーンにインストールされます。



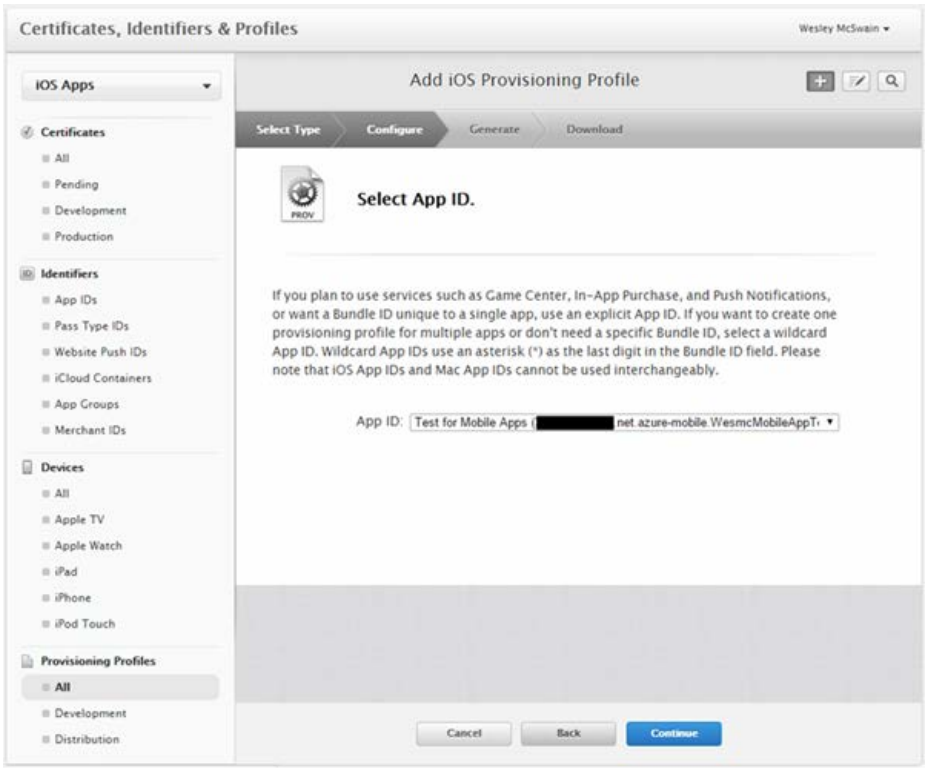
> **メモ:**
>
> 証明書の名前は異なることがありますが、名前の前に **Apple Development iOS Push Services:** が付けられます。

10. キーチェーン アクセスの [証明書] カテゴリで、作成したばかりの新しいプッシュ証明書を右クリックします。[書き出す] をクリックし、ファイルに名前を付けて、.p12 形式を選択します。次に、[保存] をクリックします。
11. エクスポートした .p12 証明書のファイル名と場所を記録します。これは、Azure クラシック ポータルにアップロードして APNS との認証を有効にするために使用されます。

アプリケーションのプロビジョニング プロファイルを作成する

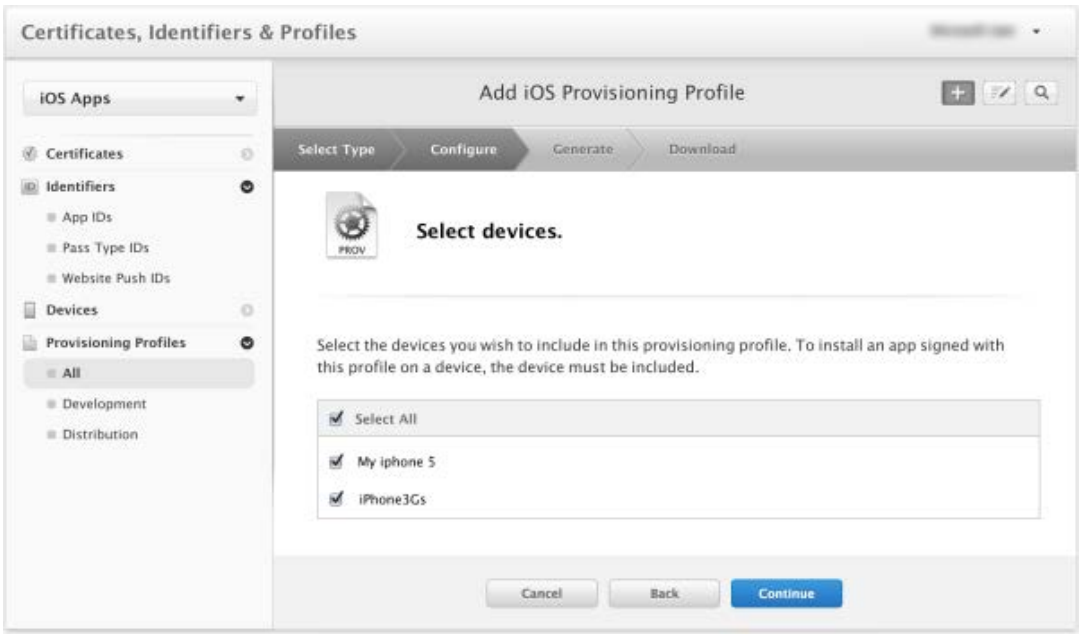
1. iOS Provisioning Portal に戻って [Provisioning Profiles] を選択し、[All] を選択してから + ボタンをクリックして、新しいプロファイルを作成します。これで、Add iOS Provisiong Profile ウィザードが起動されます。

2. [Development] でプロビジョニング プロファイルの種類として [iOS App Development] を選択し、[Continue] をクリックします。
3. 次に、[App ID (アプリ ID)] ドロップダウン リストから作成したばかりのアプリケーション ID を選択し、[Continue (続行)] をクリックします。

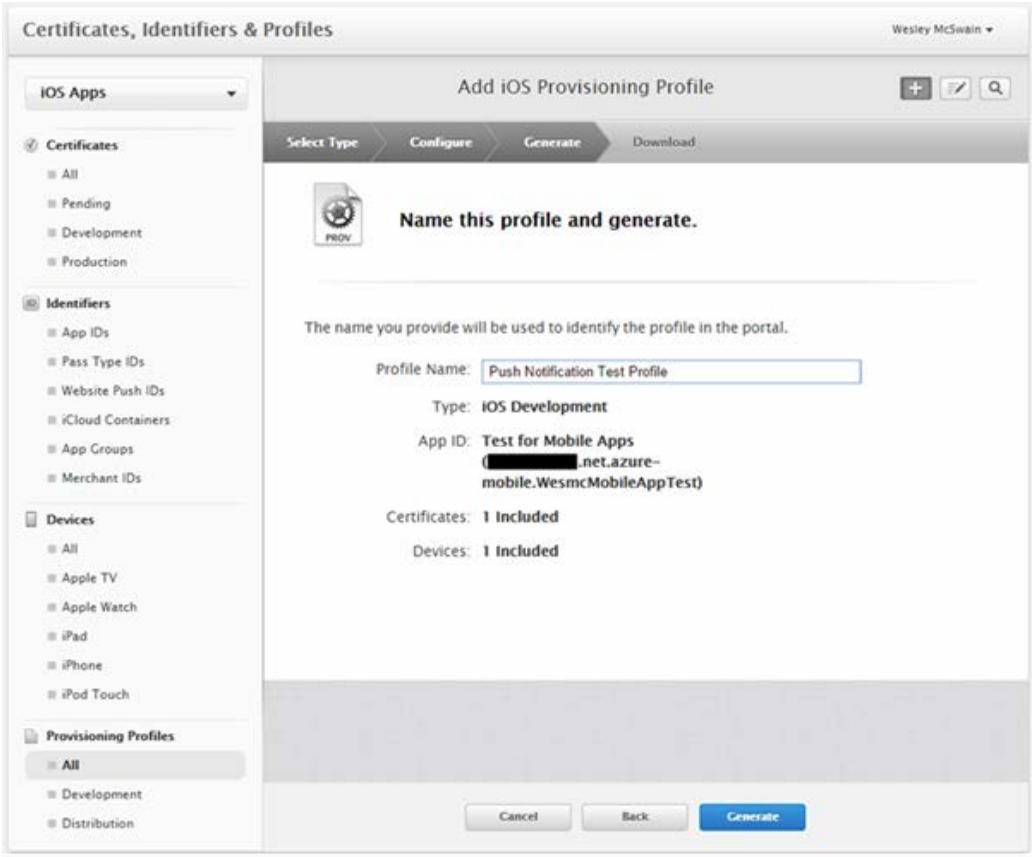


4. [Select certificates (証明書の選択)] 画面で、コードの署名に使用される開発証明書を選択して、[Continue (続行)] をクリックします。これは署名証明書で作成したプッシュの証明書ではありません。

5. 次に、テストに使用するデバイスを選択し、[Continue] をクリックします。

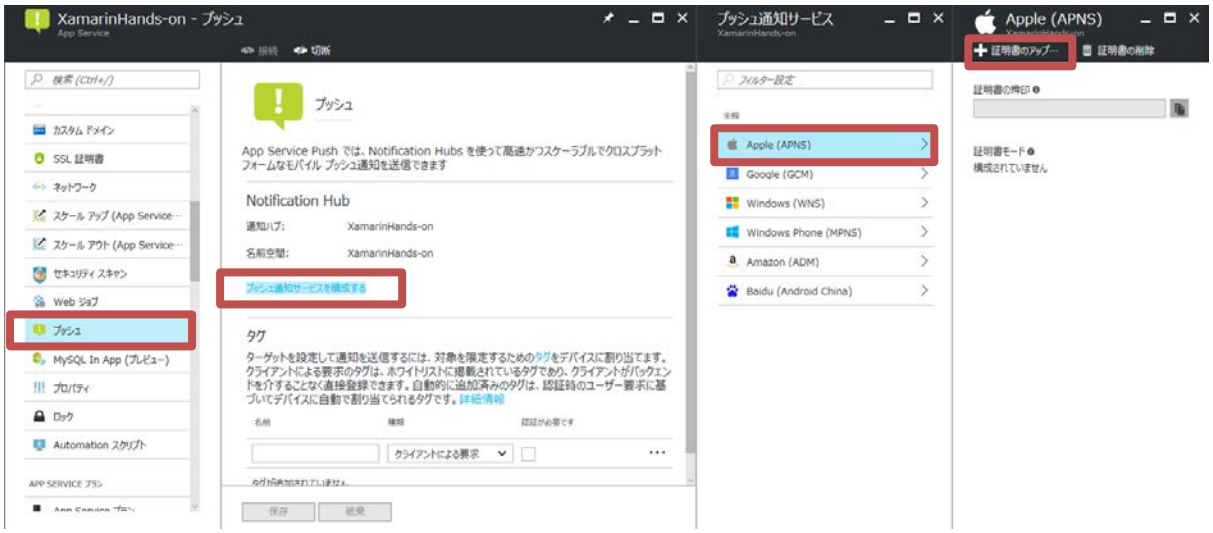


6. 最後に、[Profile Name] でプロファイルの名前を選択し、[Generate] をクリックします。



APNS 用に通知ハブを構成する

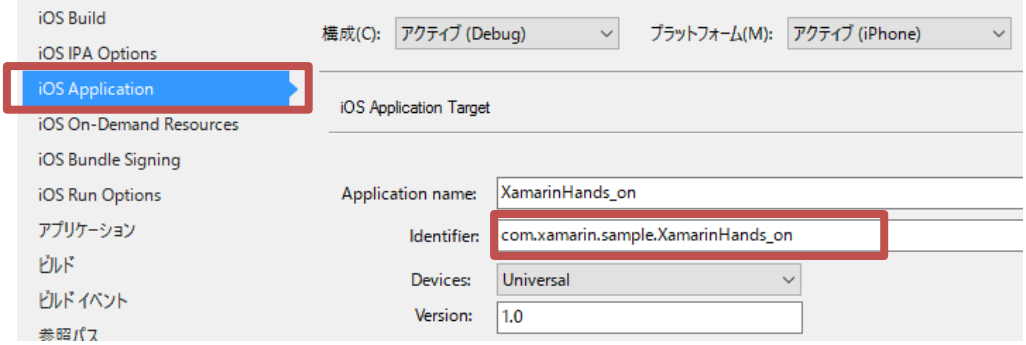
1. Azure ポータル にログオンし、アプリケーションに移動します。次に、[設定] の [プッシュ] をクリックします。続いて、[プッシュ通知サービスを構成する] をクリックします。[プッシュ通知サービス] ブレードが表示されるので、[Apple (APNS)] をクリックし、[証明書の拇印]に 以前にエクスポートした .p12 プッシュ証明書ファイルをアップロードします。



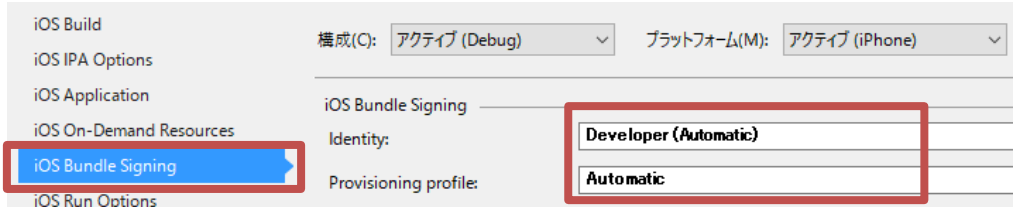
これで、iOS のプッシュ通知と連携するようにサービスが構成されました。

iOS プロジェクトの構成

1. Visual Studio で、iOS プロジェクトを右クリックし、[プロパティ] をクリックします。
2. [プロパティ] ページで、[iOS Application] タブをクリックし、前に作成した ID で [Identifier] を更新します。



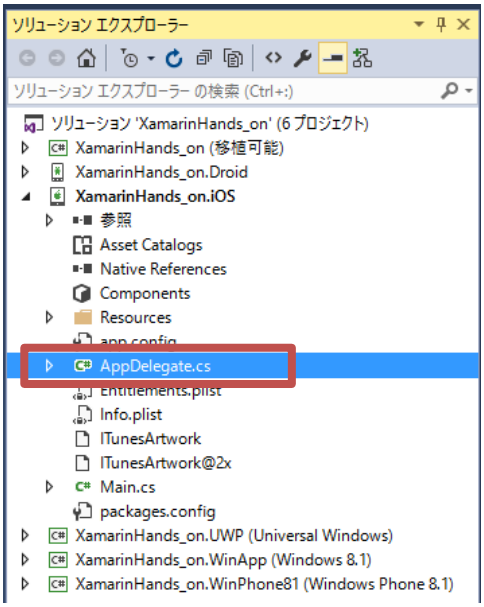
3. [iOS Bundle Signing (iOS バンドル署名)] タブで、対応する [Identity] とこのプロジェクトに対して設定した [Provisioning profile] を選択します。



これで、プロジェクトではコード署名のために新しいプロファイルを使用するようになります。公式の Xamarin デバイス プロビジョニングのドキュメントについては、Xamarin デバイス プロビジョニングに関するページを参照してください。

iOS アプリへのプッシュ通知の追加

1. iOS プロジェクトで AppDelegate.cs を開きます。



2. 次の using ステートメントをコード ファイルの先頭に追加します。

更新前

AppDelegate.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
```

更新後

AppDelegate.cs

```
using Newtonsoft.Json.Linq;
using Microsoft.WindowsAzure.MobileServices;

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
```

3. AppDelegate クラスで、通知に登録するために RegisteredForRemoteNotifications イベントのオーバーライドを追加します。

更新前

AppDelegate.cs

```
public partial class AppDelegate :
global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
{
    // ・・・・省略・・・・
```

```
}
```

更新後

AppDelegate.cs

```
public partial class AppDelegate :
global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
{
    // ・・・・省略・・・・

    public override void RegisteredForRemoteNotifications(
                                    UIApplication application, NSData
deviceToken)
    {
        const string templateBodyAPNS =
                            "{\"aps\":{\"alert\":\"$(messageParam)\"}}";

        JObject templates = new JObject();
        templates["genericMessage"] = new JObject
        {
            {"body", templateBodyAPNS}
        };

        // Register for push with your mobile app
        Push push = TodoItemManager.DefaultManager.CurrentClient.GetPush();
        push.RegisterAsync(deviceToken, templates);
    }
}
```

4. AppDelegate に次の DidReceivedRemoteNotification イベント ハンドラーのオーバーライドも追加します。

更新前

AppDelegate.cs

```
public partial class AppDelegate :
global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
{
    // ・・・・省略・・・・
}
```

更新後

AppDelegate.cs

```
public partial class AppDelegate :
global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
{
    // ・・・・省略・・・・
```

```
    public override void DidReceiveRemoteNotification(
        UIApplication application, NSDictionary userInfo,
        Action<UIBackgroundFetchResult> completionHandler)
    {
        NSDictionary aps = userInfo.ObjectForKey(new NSString("aps")) as
NSDictionary;

        string alert = string.Empty;
        if (aps.ContainsKey(new NSString("alert")))
            alert = (aps[new NSString("alert")] as NSString).ToString();

        //show alert
        if (!string.IsNullOrEmpty(alert))
        {
            UIAlertView avAlert = new UIAlertView("Notification", alert, null,
"OK", null);
            avAlert.Show();
        }
    }
}
```

このメソッドは、アプリの実行中に受信した通知を処理します。

5. AppDelegate クラスの FinishedLaunching メソッドに次のコードを追加します。

更新前

AppDelegate.cs

```
public override bool FinishedLaunching(UIApplication app, NSDictionary
options)
{
    // Initialize Azure Mobile Apps
    Microsoft.WindowsAzure.MobileServices.CurrentPlatform.Init();

    // Initialize Xamarin Forms
    global::Xamarin.Forms.Forms.Init();

    LoadApplication(new App());

    return base.FinishedLaunching(app, options);
}
```

更新後

AppDelegate.cs

```
public override bool FinishedLaunching(UIApplication app, NSDictionary
options)
{
    // Initialize Azure Mobile Apps
```

```
    Microsoft.WindowsAzure.MobileServices.CurrentPlatform.Init();

    // Initialize Xamarin Forms
    global::Xamarin.Forms.Forms.Init();

    LoadApplication(new App());

    // Register for push notifications.
    var settings = UIUserNotificationSettings.GetSettingsForTypes(
                UIUserNotificationType.Alert
                | UIUserNotificationType.Badge
                | UIUserNotificationType.Sound,
                new NSSet());

UIApplication.SharedApplication.RegisterUserNotificationSettings(settings);
    UIApplication.SharedApplication.RegisterForRemoteNotifications();

    return base.FinishedLaunching(app, options);
}
```

これによってリモート通知のサポートが有効になり、プッシュ登録の要求が行われます。

これで、アプリケーションがプッシュ通知をサポートするように更新されました。

iOS アプリでプッシュ通知をテストする

1. iOS プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. Visual Studio で [実行] または F5 キーを押してプロジェクトをビルドし、iOS デバイスでアプリを起動します。[OK] をクリックして、プッシュ通知を受け入れます。

   メモ

   アプリケーションからのプッシュ通知を明示的に受け入れる必要があります。これが必要であるのは、初めてアプリケーションを実行するときだけです。

3. アプリケーションで、タスクを入力し、プラス (+) アイコンをクリックします。
4. 通知が受信されたことを確認し、[OK] をクリックして通知を破棄します。

関連リンク

- Microsoft Azure Mobile Apps
  https://azure.microsoft.com/ja-jp/services/app-service/mobile/
- Visual Studio Dev Essential
  https://www.visualstudio.com/ja/dev-essentials/
- Visual Studio - Xamarin クロスプラットフォーム開発
  https://www.microsoft.com/ja-jp/dev/campaign/vs-xamarin.aspx
- 公式 Xamarin ドキュメント(英語)
  https://developer.xamarin.com/guides/
- 無料版「Creating Mobile Apps with Xamarin.Forms」電子書籍 (英語)
  https://developer.xamarin.com/guides/xamarin-forms/creating-mobile-apps-xamarin-forms/
- Visual Studio - Xamarin クロスプラットフォーム開発
  https://www.microsoft.com/ja-jp/dev/campaign/vs-xamarin.aspx
- 公式 Xamarin ドキュメント(英語)
  https://developer.xamarin.com/guides/
- 無料版「Creating Mobile Apps with Xamarin.Forms」電子書籍 (英語)
  https://developer.xamarin.com/guides/xamarin-forms/creating-mobile-apps-xamarin-forms/
- Visual Studio と Xamarin
  https://msdn.microsoft.com/ja-JP/library/mt299001.aspx
- Xamarin Releases
  https://releases.xamarin.com/
- Xamarin Open Source
  http://open.xamarin.com/

- Microsoft エンタープライズ向け Premier サポート
  https://www.microsoft.com/ja-jp/services/premier.aspx


Microsoft