Developer notes

Solutions and projects

- `VPCConnect.sln` - core VPC Connect application
  - `ConnectApp` - shared UI, implemented with Xamarin.Forms
  - `ConnectApp.Android` - Android native app
  - `ConnectApp.iOS` - iOS native app
  - `ConnectApp.AppTests` - tests for the apps
- `VPCConnectSmokeTests.sln` - smoke tests for the app
  - `PortalApi.Tests` - tests for the portal and API

See also: Build notes

Secrets

The Android app signing keystore and password, and the required iOS development certificates and distribution profiles are not included in this repository.

As mentioned in the API notes, the portal API requires an access key to accompany all requests. This is not included in the repository.

Provide class `SensitiveConstants` in `ConnectApp/Communication/SensitiveConstants.cs`

```
namespace ConnectApp.Communication
{
    public class SensitiveConstants
    {
#if TESTPORTAL
        public static readonly string PortalApiAccessCode = "<Portal API access code>";
        public static string Tests_Username = "<username for testing>";
        public static string Tests_Password = "<password for testing>";
        public static string Tests_FullName = "<full name for testing>";
        public static string Tests_SignedInUrl = PortalUris.PortalWeb_BaseUri;
#else
        public static readonly string PortalApiAccessCode = "<Portal API access code>";
        public static string Tests_Username = "<username for testing>";
        public static string Tests_Password = "<password for testing>";
        public static string Tests_FullName = "<full name for testing>";
        public static string Tests_SignedInUrl = PortalUris.PortalWeb_BaseUri;
#endif
    }
}
```

Android/iOS projects

You shouldn't need to touch much in the Android or iOS native apps - they are effectively wrappers for the ConnectApp project, which defines the UI and does the work of completing registration and handling push notifications.

Key resources in the native apps:

- `ConnectApp.Android/google-services.json` - downloaded from FCM, and must have a build type of: `GoogleServicesJson`
- `ConnectApp.iOS/GoogleService-Info.plist` - downloaded from FCM, and must have a build type of: `BundleResource`

Key modifications in the native apps are marked out by comments starting with: `// FCM:`

Startup

The core of the application starts up in `ConnectApp/App.xaml.cs`.

Here a handler for `CrossFirebasePushNotification.Current.OnTokenRefresh` registers for the push notification token (when initially provided, or refreshed). On receipt of the token, the app then submits it to the portal through `Communication/PortalApi.SubmitPortalDeviceCheckAsync` to check whether the app is already registered.

If not registered, the Home page prompts the user to switch to the Connection page.

When viewing the Connection page:

- If the registration is found, the UI switches immediately to state `PortalRegisterComplete`.
- If not, the UI switches to state `FormReady` - showing a username and password field for the user to complete.

On completion of the form, the app then:

- Exchanges username and password for a user token, with: `Communication/PortalApi.GetUserTokenAsy`
- Registers, passing its push token, user token, and device details to: `Communication/PortalApi.SubmitPortalRegistrationAsync`
- If the registration is accepted, the UI switches to state `PortalRegisterComplete`.
- If not, it switches to state `PortalRegisterFail` - showing the form again for the user to try again.

UI

- UI classes and xaml are available in `ConnectApp/Pages`
- Pages extend `BaseAppContentPage`.
- Some enums are translated to human readable text using classes in `ConnectApp/Text`.

Database

The app is supported by a SQLite database, and classes to manage it are found in:

- `ConnectApp/Database` - database management classes
- `ConnectApp/Entities` - classes that map to database tables

The database is defined in `ConnectApp/Database/ConnectDb.cs`.

If any of the entities or the database definition changes, increment `ConnectApp/Database/DbConstants.Version`. This will cause the database to be rebuilt the next time the app is opened.

API

See also: API notes

Communication with the portal API is handled by classes in `ConnectApp/Communication` and `ConnectApp/DTO`.

- `PortalUris.cs` contains details of the various portal endpoints. These are switched based on the current build environment.
- `PortalApi.cs` contains a number of methods able to call the portal endpoints and return the response.

Logging

All logging is managed by the `AppLogger` class in `ConnectApp/AppLog`.

When logging, call one of the public methods:

- `Verbose(string message, bool sensitive, Exception exception)`
- `Debug(string message, bool sensitive, Exception exception)`
- `Info(string message, bool sensitive, Exception exception)`
- `Warning(string message, bool sensitive, Exception exception)`
- `Error(string message, bool sensitive, Exception exception)`

- `Exception(Exception exception)`

For regular logs, you must set the `sensitive` parameter to indicate whether the log could contain sensitive information (such as push tokens, user tokens, or other credentials).

Sensitive information will not be passed to the system log (viewable through adb logs) in production builds, but is available in debug builds.

NuGet packages

The following packages are used across the solution:

- Firebase Push Notification plugin, CrossGeeks
- RestSharp (a RESTful HTTPS library)
- sqlite-net-pcl (SQLite database)
- Xamarin.Forms (shared UI library)
- Xamarin.Essentials (shared device tools)
- CsvHelper (CSV library)
- Newtonsoft.Json (JSON library)
- Xam.Plugins.Logging (logging support)
- Xamarin.Firebase.Core
- Xamarin.Firebase.Analytics
- Xamarin.Firebase.Crashlytics
- Xamarin.Firebase.iOS.Analytics
- Xamarin.Firebase.iOS.Crashlytics