Characteristics of a Native Mobile App project in WEM.

WEM supports Android with API level 21 and higher. This is Android Lollipop 5.0 as lowest/oldest version.

Understand the differences between a standard web-portal and a native app:

- A webportal is centralized, running at a single location in a cloud on heavy servers with high availability, lots of diskspace, CPU power and internet-bandwidth;
- A native app is running on a small personal device, on multiple unknown locations, limited diskspace, limited cpu-power and limited or no connection at all.

- A webportal is one website to serve many users (in general), so specific users could/should be authenticated in some way (basic authentication, Single Sign On using ADFS or oAuth2);
- A native app is running on a personal device, which can in itself be regarded as a trusted single user situation. A different way to authenticate the user should be considered – like a one-time authentication with an online/external authentication provider, after which a specific key with optionally a limited validity period is stored on the device so next time the app is started, the authentication can be skipped because the device and local storage are trusted.

- A webportal can be published from WEM and be available to end-users straight away.
- A native app can be generated in WEM, but the distribution (getting the app onto the devices) is still a separate process which require manual actions. The result which is generated by WEM is the native mobile equivalent of an installer file. There are several ways to get the file onto a single device or distributed to many devices, different for Android and iOS.

- A webportal has internet connectivity. Many widgets currently in WEM (like charts and google maps) are created with web and connectivity in mind. Many of those widgets might not work on native mobile devices or only when a data connection is available.

WEM features that will **not** work on native mobile devices:

- OData will not work and will not be supported on mobile, the only ways to exchange data between an app on mobile devices is by using rest or soap services.
- Authentication providers based on SAML like ADFS for Corporate Single Sign on will not (easily) work, as they require a trusted endpoint at a reachable and known destination. A mobile device is not. OAuth using Google, Facebook and those more personal authenticators can be used.
- Comet for real time messaging will not work (it requires connectivity and known endpoints); Firebase messaging is needed for native mobile. Support is not yet available in WEM.
- Widgets requiring data connection (like charts, google maps, recaptcha) will only work when data connectivity is available, because they require resources that are external, not part of the app itself (like map data of the google javascript libraries that are available online). Specific widgets might be developed with web in mind, so just be careful which ones you use. WEM Modeler will provide means to add information about which platforms a specific

widget supports (web, android, ios). How this is going to work in reality, is yet to be made clear…

- Incoming calls, requests, services cannot work (as the address, location and connectivity of a mobile device is unknown). So a mobile app can only feature consuming webservices and consuming http requests.
- Very deeply normalized data structures with many references should be avoided:
    - In SQL Server, backing the web portals, normalized datastructures with references is not an issue, querying that datastore will use the SQL join structures and optimization features and can rely on the power of high-end database servers;
    - The local device storage does not have the SQL features for joining and query optimizations, so it is better to have the data as "flat" as possible, targeted at the functionality for the mobile device, and if this means the data should be de-normalized (and might thus be duplicated), support that in the specific wem models and flows, to make queries faster.
- Local files on device filestore: limited, but yes, local files are accessible (when using a file input you can select local files which are public, like pictures and documents in some folders).

Keep the mobile app project small

A project in WEM contains all kinds of assets. Flowcharts, templates, items in ontology, items in the file library, icons and more. All these assets will be contained in the file that is generated as native mobile app. So they will add KBs or even MBs to the file. Keep your mobile project as clean as possible, remove all assets you do not need in the app. That's also one reason to have your mobile app as a separate WEM-project, not part of a larger projects that also contains many webportals with many assets which are not used in the native mobile app (currently, there is a strict separation on project level: you need to make a choice when creating a project whether it will be a web or a mobile project). If you do go down the path to have all kinds of portal types (both web and mobile) in one project, be aware of the effects of the resulting file size.

If you finally have an apk-file for android, you can rename the file locally to a .zip file and extract its contents, to see how large the file is and how much certain resources are. The extracted folder will contain a folder assets, in there you'll find the portal folder. This folder contains all specific WEM project/portal resources (all other resources in the apk you should regard as fixed, same for every other build).

Synchronizing data

A typical situation might be:

- You have a web portal/project for all features, including user management, authentication, data management, administrator features.
- You want to have native mobile apps that have basically a part of the end-user features of the web, without all the management and administrator features. This app should be able to work offline but still have the necessary data. Some (not all) changes made on the device must be made available on the centralized web as well.

To make this work, you'll need to make data synchronization features, by exposing webservices from the web portal (soap and/or rest) and consuming them in the native app. Based on the data and the features your app should support, you will need to figure out a synchronization strategy: add extra fields for version and last updated timestamps, decide which datastore is leading (usually the central web for application data and the local device for personal data and preferences for working with the app), and how nitty-gritty you want to sync. It is easier to just clear all application data on device and replace it with all data from web (and only do this if user request to force complete update, or when you can see that there are changes available on the web which are not yet available on the device, by your own means of checking this). Work out the best scenarios and use several update calls. Small datasets can best be transferred using REST (easiest to model in WEM, using the json import/export nodes). Large datasets can be transferred using SOAP, this is more work in the Modeler, but gives you the option to do some extra work in between, like showing progress bars to indicate how much data has been processed and if necessary, perform some data manipulation and transformation (but keep in mind: as little as possible on the device-consumer, and as much as possible on the web-provider). Please understand that there is no magic automatic synchronisation and that all data interchange between a web portal (or any other source) and the native mobile app must be custom made in WEM.

Some possibly useful highlights

WEM Keywords

https://my.wem.io/forum?threadid=145 about useful keywords for mobile:

- ConnectivityTypes
- Platform
- BatteryLife
- DeviceId
- DeviceName

If you want to check whether a user has data connectivity (which on a native mobile device, can be unavailable), you can check:

```
ConnectivityTypes contains "wifi"
or
ConnectivityTypes contains "mobile"
or
ConnectivityTypes contains "ethernet"
```

And use this in a Calculated Field.

UI

- Use the available Mobile Android or Mobile iOS master templates to get close to native look and feel.
- Limit the use of overlays, use mostly full pages and use (and check using xs and sm in the template editor) the available options for responsiveness.
- Use png-images as much as possible for icons, logos etc, especially in the app settings, icons, splash screen.
- Limit the use of navigation items in menus.

Synchronize Data

Native mobile apps can use local data storage which can be useful when there is no data connection available. A freshly installed app on device will not contain any data, you need to manage it in the flows, based on whatever you want and need to do in the app.

Use webservices when a data-connection is available:

- REST/json is fastest with lowest overhead and footprint;
- SOAP/Xml can sometimes be more useful when the amount of data is large and needs to be processed in batches with progress-info;
- In the WEM web-portal which is the datasource, create webservices specifically for mobile to keep the client-footprint as small as possible. Any service method exposed in WEM can be used via SOAP and REST, you can make the choice in the app as you deem best. (I've used both, where if one flow using SOAP failed, another try with REST was done…)
- Use checks using dates for last modified and last synchronized to make a lightweight check to decide if some parts of local data need to be synchronized. Perform this check on start of the mobile app if there is a data connection available, then show user information if data has to be synchronized (if from tests you know that it takes time, let user decide to perform the sync).
- If data does not have to be accessible while offline, and tests show that direct usage via webservice over active data connection is viable, go with that.
- If user data is stored, linked to a user account, also provide a way where a user can clear the local data and profile completely on the device, but also on the web portal (in case device gets lost). You can use the DeviceID for this purpose, a specific Token value with optionally a valid period, and store this with the user's account in the data source web portal, which the user can clear and next time the app checks the useraccount, check if this DeviceId and Token are still valid. Or have a list of DeviceIds and tokens, if user can have multiple devices linked.

Apple Development

Distribution of iOS apps.

Create certificates and provisioning profiles in developer portal: https://developer.apple.com/

A link to this portal is also available in the iOS Provisioning and Certificates overlay (accessible via Portal Settings).

For distribution of an iOS app, one needs to have at least a Developer account with Apple, create App IDs, create certificates and provisioning profiles to link to those App IDs, add owned and trusted devices on which apps are allowed to be installed for testing purposes.

All these parts (App ID, Certificates, Provisioning profiles, linked devices) are all linked and need to be correctly set up in the Apple Developer Environment.

In WEM, the App ID should be the same as defined in the Apple Developer environment.

In WEM, a certificate request should be started (if you do not have a proper certificate already), the request should be uploaded to Apple at the creation of a certificate, resulting in a certificate to download, which should be uploaded into WEM. There are different types of certificates, at least you need one for Development and one for Distribition.

Then, the provisioning profiles can be made. One for development and one for distribution.

The development provisioning profile is where you can link one or a few existing iOS devices linked to your Development account. These devices can be used to test the app via the OTA in WEM iOS Build overlay.

The distribution provisioning profile is where you can decide to provision it for the public Apple App Store (after approval by Apple), or if you will be using a limited set of registered devices.

A few links:
https://support.magplus.com/hc/en-us/articles/204270188-iOS-Creating-an-Ad-Hoc-Distribution-Provisioning-Profile
https://developer.apple.com/programs/
Create an Apple Developer Account, with your (or a company) AppleID, join the Apple Developer Program

Create certificates and provisioning profiles in developer portal: https://developer.apple.com/

Here you define the bundleId which is also used in the WEM iOS project, they should match on bundleId and linked certificate and provisioning profiles.

Wildcard is possible, if you are planning to have more apps within the same provisioning profile and certificate, you just define a bundle id with a common start like **io.mycompanyname.myiosapps.***

Use these certificates and provision profiles (download from developer portal and import into WEM with your iOS App Project).

Make sure you use the correct bundle id (or with a wildcard bundle use the proper prefix, so the certificate and provisioning are valid for this bundle).

Build app in WEM (like publish, but then slightly different, creating the installer file [appname.ipa], currently on the Settings page in WEM v4)

Use the WEM OTA-option to load it onto your iOS testing device(s) which is/are linked to your App Developer Account (developer.apple.com portal)

App Store Connect Documentation:

Manage all required agreements, contacts, banking and tax information in the appstore connect portal.
https://help.apple.com/app-store-connect
https://appstoreconnect.apple.com/

Add the app to your AppStore Connect environment,

https://help.apple.com/app-store-connect/#/dev2cd126805

Fill in all required information.

Continue to enter app details

https://help.apple.com/app-store-connect/#/dev97865727c
https://developer.apple.com/business/custom-apps/
https://help.apple.com/app-store-connect/#/dev275598f16
https://help.apple.com/businessmanager/

Permission Purpose Descriptions

Before sending in to Apple Store for admission, **WEM needs to manually set the following** information regarding the purpose of the permissions for Camera and Location.

(pages to manage this in Modeler is work in progress)

These texts will be scrutinized by Apple, they need to be specific and correct, not a standard content.

```
{
  PortalPermissions:[
   {
   PortalId: ....,
   NSCameraUsageDescription: "This app does not use your camera.",
   NSPhotoLibraryUsageDescription: "This app does not access your photo library.",
   NSLocationWhenInUseUsageDescription: "Your location is used to show our organizations in your vicinity and to calculate the distance to these organizations from your location.",
   NSLocationAlwaysUsageDescription: "Your location is only used when you are actively looking for organizations in your vicinity."
   }
  ]
}
```