Building custom apps using the SugarCRM mobile SDK

An UnCon 2017 tutorial

Greg Khanlarov, gkhanlarov@sugarcrm.com
Andrew Zhukovsky, azhukovsky@sugarcrm.com
Vitaly Makarevich, vmakarevich@sugarcrm.com

Sugar v7.9.1.0
Mobile SDK 6.0.0.X-1.2.0-1

Get a digital copy of this tutorial at http://bit.ly/tutorial_mobilesdk

Table of contents

Introduction

The SugarCRM mobile app is packed with features. The Mobile Application Configuration Service (MACS) comes in handy when you need to build a branded version of Sugar's standard mobile offering. But what if you need to add custom functionality? Not to worry! SugarCRM Mobile SDK comes to the rescue. In this tutorial, we will use the SugarCRM Mobile SDK to not only brand and style the mobile app but also add a custom view that renders a map with meetings locations.

What you'll learn

After completing this tutorial, you will know how to…

- Generate a custom mobile app.
- Perform basic customizations like rebranding the mobile app.
- Implement a simple custom view.
- Add a custom geolocation record action.
- Integrate the native Google Maps plugin.

Before you begin

Before we kick things off, make sure you have done each of the following:

- Have a locally running version of Sugar with the superhero sample data and modules installed (see instructions at http://bit.ly/professorm).
- Installed the HeroLocation package from http://bit.ly/herolocation (the same way as you customized Sugar with ProfM package).
- Downloaded and installed the SugarCRM mobile SDK from http://mobiletools.sugarcrm.com/#/download-sdk by following the instructions in the Getting Started Guide. **Note:** You need to have a sugarcrm.com account to access the mobile portal.

IMPORTANT: Each assignment below has a set of files and custom code that you will integrate into your custom app. This code is located in the GitHub repo **mobile_sdk/snippets** folder at http://bit.ly/mobilesdk_snip. You can find the complete solution in the Uncon GitHub repository at http://bit.ly/mobilesdk_hero.

Assignment 0: Generate the custom mobile app

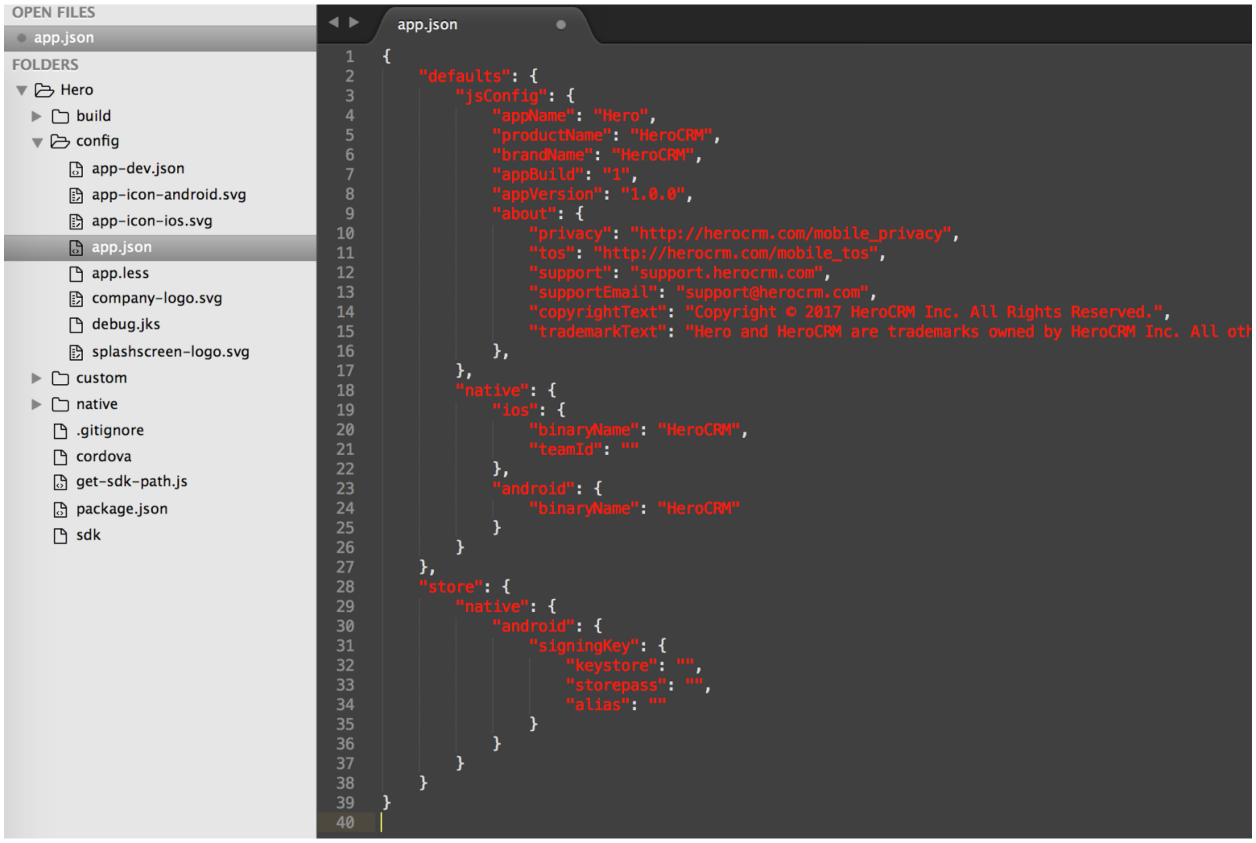
Great! Now that you have the mobile SDK installed, we can start playing with it. Let us generate your first custom mobile app. Open a terminal window and type the following command:

➔
```
$SUGAR_MOBILE_SDK_HOME/[sdk-version]/sdk/generate-app [app-folder]
```

Replace [sdk-version] with the version of SDK you have installed and the [app-folder] with the directory name where you want the app to be generated. The script will ask you for:

- App name: Enter `Hero`.
- Sugar instance URL: Enter the URL of your ProfessorM server (for example `http://localhost:8888/profm`).
- "Copy sample code?": Answer `No`.
- "Generate Android release key": Answer `No`.

Your custom app folder structure and configuration file should look like the following after you have generated the app:



```
{
    "defaults": {
        "jsConfig": {
            "appName": "Hero",
            "productName": "HeroCRM",
            "brandName": "HeroCRM",
            "appBuild": "1",
            "appVersion": "1.0.0",
            "about": {
                "privacy": "http://herocrm.com/mobile_privacy",
                "tos": "http://herocrm.com/mobile_tos",
                "support": "support.herocrm.com",
                "supportEmail": "support@herocrm.com",
                "copyrightText": "Copyright © 2017 HeroCRM Inc. All Rights Reserved.",
                "trademarkText": "Hero and HeroCRM are trademarks owned by HeroCRM Inc. All oth
            },
        },
        "native": {
            "ios": {
                "binaryName": "HeroCRM",
                "teamId": ""
            },
            "android": {
                "binaryName": "HeroCRM"
            }
        }
    },
    "store": {
        "native": {
            "android": {
                "signingKey": {
                    "keystore": "",
                    "storepass": "",
                    "alias": ""
                }
            }
        }
    }
}
```

Open `config/app-dev.json` and ensure the `siteUrl` points to your your Sugar instance. If you don't have `config/app-dev.json`, you can create one and add the following:

```
{
  "jsConfig": {
    "siteUrl": "http://your-sugar-instance"
  }
}
```

Now we will play with the SDK command line tools. In a shell, navigate to your custom app folder and type the following commands:

- ➔ `./sdk help`
- ➔ `./sdk debug`
- ➔ `./sdk build -p android`
- ➔ `./sdk build -p ios`

The first command prints out help information about SDK build tools. The second runs a livereload debug server, which allows you to debug the mobile web app in a desktop browser. Many customizations can be played with in a desktop environment! Open the web app by typing `http://localhost:9000/app` in Chrome or Safari. You should see the app login page that displays the Acme placeholder logo.

The last two commands build native apps for Android and iOS respectively. Depending on which native environment you set up, you may use one or the other or both by passing the `-p native` parameter.

IMPORTANT: Keep in mind that you need an active Apple Developer account with a valid developer certificate to be able to compile and install the app on an iOS device.

TIPS:

1. Turn off offline mode to minimize the number of network requests. This should simplify debugging. To turn off the offline mode, add the following code into `jsConfig` section of `config/app-dev.json.`

```
"offline": {
    "enabled": false
}
```

2. Turn off welcome help. This should also simplify debugging:

```
"ui": {
    "showTutorial": false
}
```

Make sure `app-dev.json` doesn't have syntax errors by checking for missing and extra commas.

Congrats! You have just created a custom mobile application without writing any code whatsoever! Now let's see how we can make it really custom.

Assignment 1: Brand and Style

Now let's perform some basic customizations with your mobile app. We will change app icons, splashscreen, company logo, and some colors in the UI. To start, do the following:

1. Replace all svg files in your local **`Hero/config`** folder with the ones from http://bit.ly/mobilesdk_assign1.

The SVG files are the images you will use for your app's icons and logos. The SDK build tool automatically generates a set of app icons for all required screen sizes. **`company-logo.svg`** is displayed on the app login view, and **`splashscreen-logo.svg`** is used on the app launch screen (used in native apps only). If you want to use your own logos, you need to have them in svg format.

2. Copy the contents of **`app.less`** from http://bit.ly/mobilesdk_assign1 to your local **`Hero/config/app.less`**

You can see we overrode a bunch of `@SDK_xyz` Less variables. You can control the look and feel of the app by specifying custom values for such variables. You can find the full list in the Mobile Style Guide posted at http://mobiletools.sugarcrm.com/#/download-sdk.

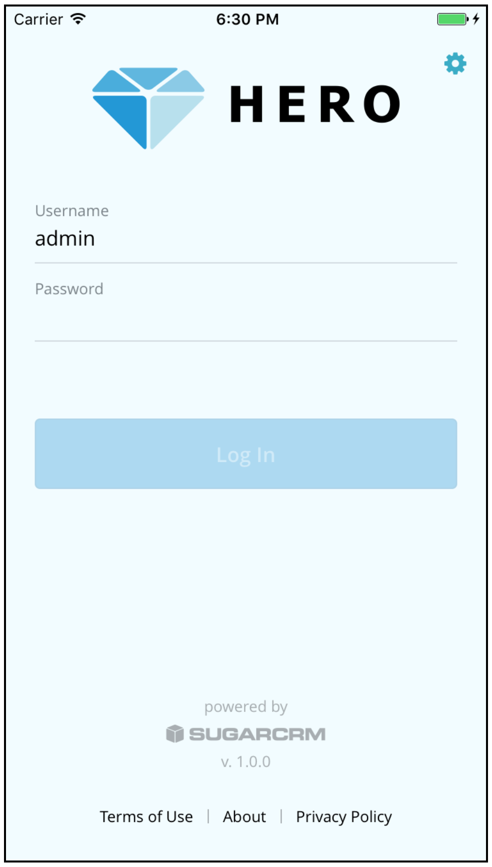
3. Update your local **`Hero/config/app.json`** with values from **`app.json`** from http://bit.ly/mobilesdk_assign1.

This will update the brand properties of the native app, specifically the iOS bundle ID and the Android package name that uniquely identify your app in AppStore.

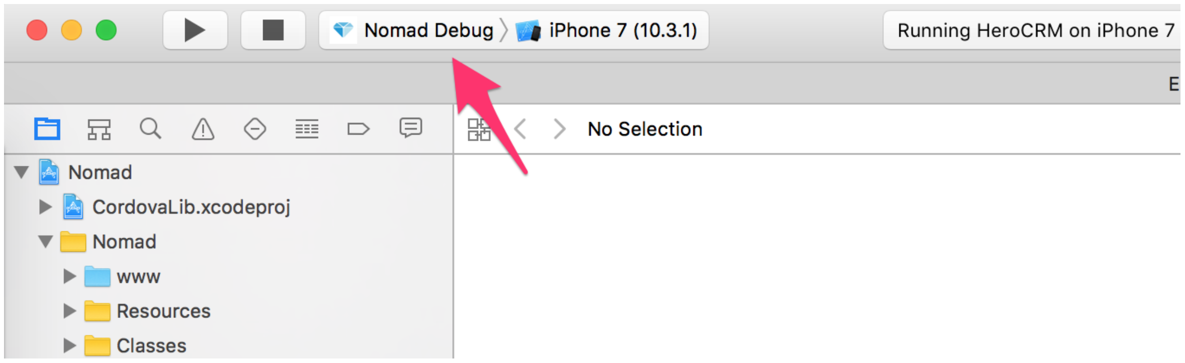
4. Run **`./sdk update-native`** to update the native projects with changes we have just made to the app configuration file.

That's it! We have just branded the app. You can test it out in a desktop browser by running `./sdk debug`. To see the changes in the native app, run `./sdk bundle-web -p native` and then launch the app from Xcode or Android Studio. You can also build binaries and install them on a device.

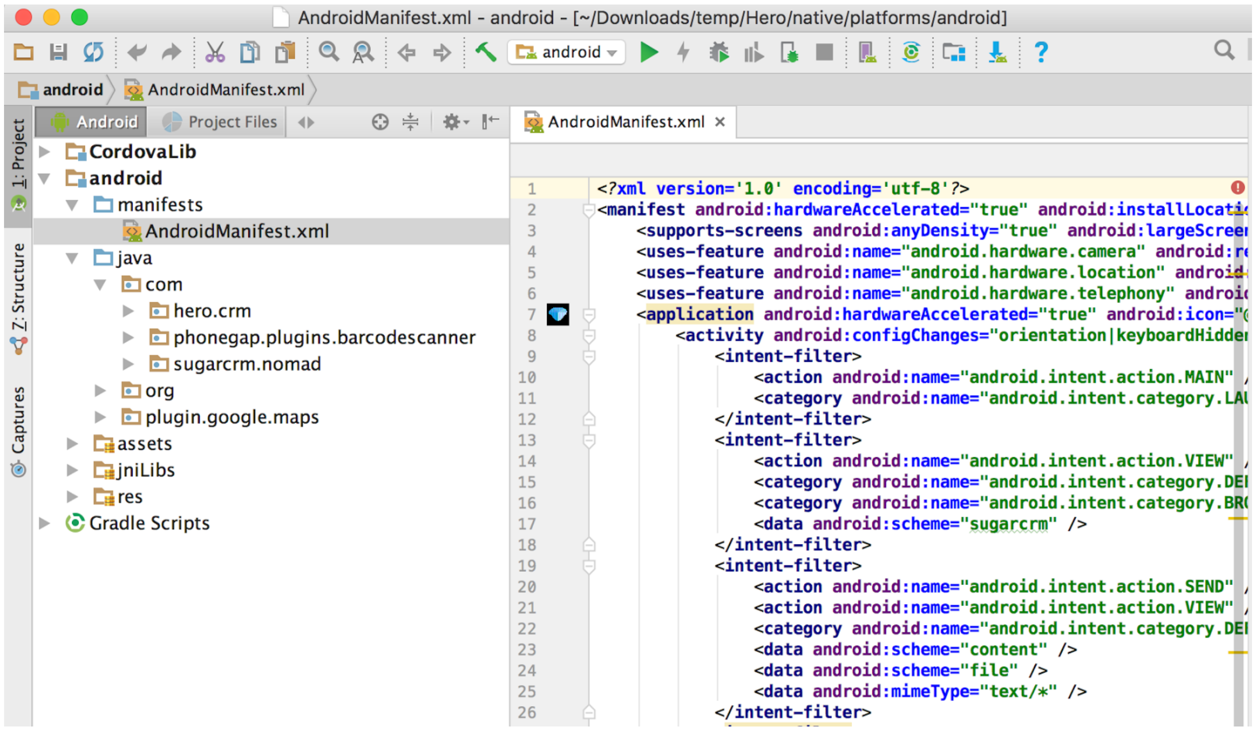
The login view should look like this one:



To run native iOS project from Xcode, navigate to `native/platforms/ios` folder and double-click `Nomad.xcodeproj` to open the project in Xcode. It's important to choose Debug scheme when running the app from Xcode:



To run the native Android app from Android Studio, open the `native/platforms/android` folder in Android Studio:



```
<?xml version='1.0' encoding='utf-8'?>
<manifest android:hardwareAccelerated="true" android:installLocati
    <supports-screens android:anyDensity="true" android:largeScree
    <uses-feature android:name="android.hardware.camera" android:r
    <uses-feature android:name="android.hardware.location" android
    <uses-feature android:name="android.hardware.telephony" androi
    <application android:hardwareAccelerated="true" android:icon="
        <activity android:configChanges="orientation|keyboardHidde
            <intent-filter>
                <action android:name="android.intent.action.MAIN" 
                <category android:name="android.intent.category.LA
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.VIEW" 
                <category android:name="android.intent.category.DE
                <category android:name="android.intent.category.BR
                <data android:scheme="sugarcrm" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.SEND" 
                <action android:name="android.intent.action.VIEW" 
                <category android:name="android.intent.category.DE
                <data android:scheme="content" />
                <data android:scheme="file" />
                <data android:mimeType="text/*" />
            </intent-filter>
```

NOTE: If you build native apps from the command line, you can find the iOS and Android binaries in `build/ios` and `build/android` folders respectively.

What did we just do? By making simple changes in the app configuration (`config/app.json`) we were able to change the color of the app header, links, and splashscreen background. We also integrated a new company logo, configured the custom app icons, and updated binary identifiers.

Let's move on and write some code.

Assignment 2: Hello World View

Branding and styling is useful, but what should you do when you need custom business logic? This section will explain how to create a simple custom view. The goal is to implement the following three things:

1. Custom view that renders "Hello, Word!" text.
2. New "hello" navigation route for the custom view.
3. "Hello" main menu action. Clicking the action will navigate to the "hello" route.

Let's begin by doing the following:

1. Create `custom/hello-world` folder. The folder "custom" is used to host all of your custom code. It's a good idea to create a separate folder for each logical piece of your business logic. We will place the Hello World view into the `hello-world` folder.

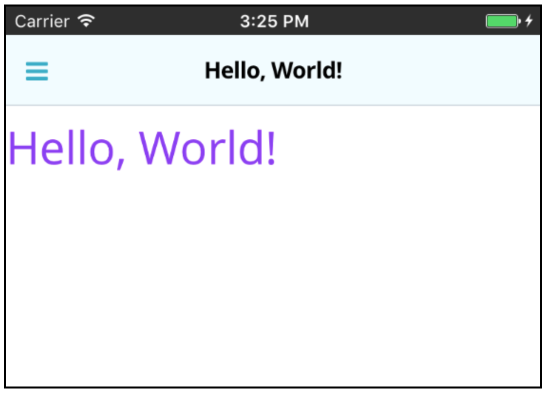
2. Copy `hello-world-view.js` and `hello-world.hbs` (both available in http://bit.ly/mobilesdk_assign2) into the `Hero/custom/hello-world` folder. Implementing a custom view can be quite simple: you declare your view class in a Javascript controller and provide a Handlebars template. Check out the contents of the files to read inline comments.
3. Copy the ui section from `app.json` from http://bit.ly/mobilesdk_assign2 into your local `Hero/config/app.json`.

Now test the app in the browser (`./sdk debug`). You should see a new custom Hello action in the main menu. Clicking it will navigate to HelloWorld view.

Let's change the color of "Hello, World!" text. If you check `hello-world.hbs`,you will notice that the text is enclosed in a span with the "`hello_world`" CSS class. Add the following code to `Hero/config/app.less`:

```
.hello_world {
  font-size: xx-large;
  color: #8a1bf6;
}
```

See the debug server automatically reload the app in the desktop browser. The text should now be purple.



Good job! You just implemented your first custom mobile view.

Assignment 3: Check-In Action

Let's have some fun with custom actions and the geolocation API! (It's the reason we installed the HeroLocation package!). Login to to your Sugar instance as an administrator and navigate to **Administration** > **Studio.** Check out the Meetings module's fields. There should be four new fields defined:

1. Check-In Time (datetime)
2. Check-In Latitude (float)
3. Check-In Longitude (float)

4. Check-In Address (text)

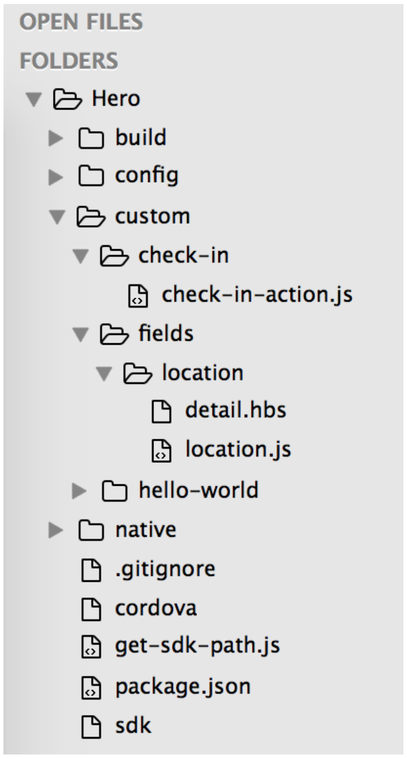
Also check out the Meeting's mobile detail layout (inside of Mobile Layouts). You should see the Location field in the list of enabled fields.

The goal of this assignment is to add a Check-In action to the Meetings record toolbar. Clicking the action will request the current location of the user's device and update the meeting with the location coordinates and timestamp. Optionally, the app will try to reverse geocode the location into a physical address.

Let's start by implementing a custom location field. The location field has a metadata type "`location`" and is a combination of latitude, longitude, and an optional address. You should put the code for your custom fields into `custom/fields/[field-type]` folder:

1. Create a `Hero/custom/fields/location` folder.
2. Copy `detail.hbs` and `location.js` from http://bit.ly/mobilesdk_assign3 and paste them in your local `Hero/custom/fields/location` folder. These two files are the field templates for the mobile detail view and javascript controller where you declare the `LocationField` class and register it for the Meetings module.
3. Create a `Hero/custom/check-in` folder.
4. Copy `check-in-action.js` from http://bit.ly/mobilesdk_assign3 and paste it in your local `Hero/custom/check-in` folder. This is the controller for the Check-In record action.
5. Copy the `location` definition from `app.json` from http://bit.ly/mobilesdk_assign3 into your local `Hero/config/app.json`. We need the icon definition because we use a custom icon for our action.

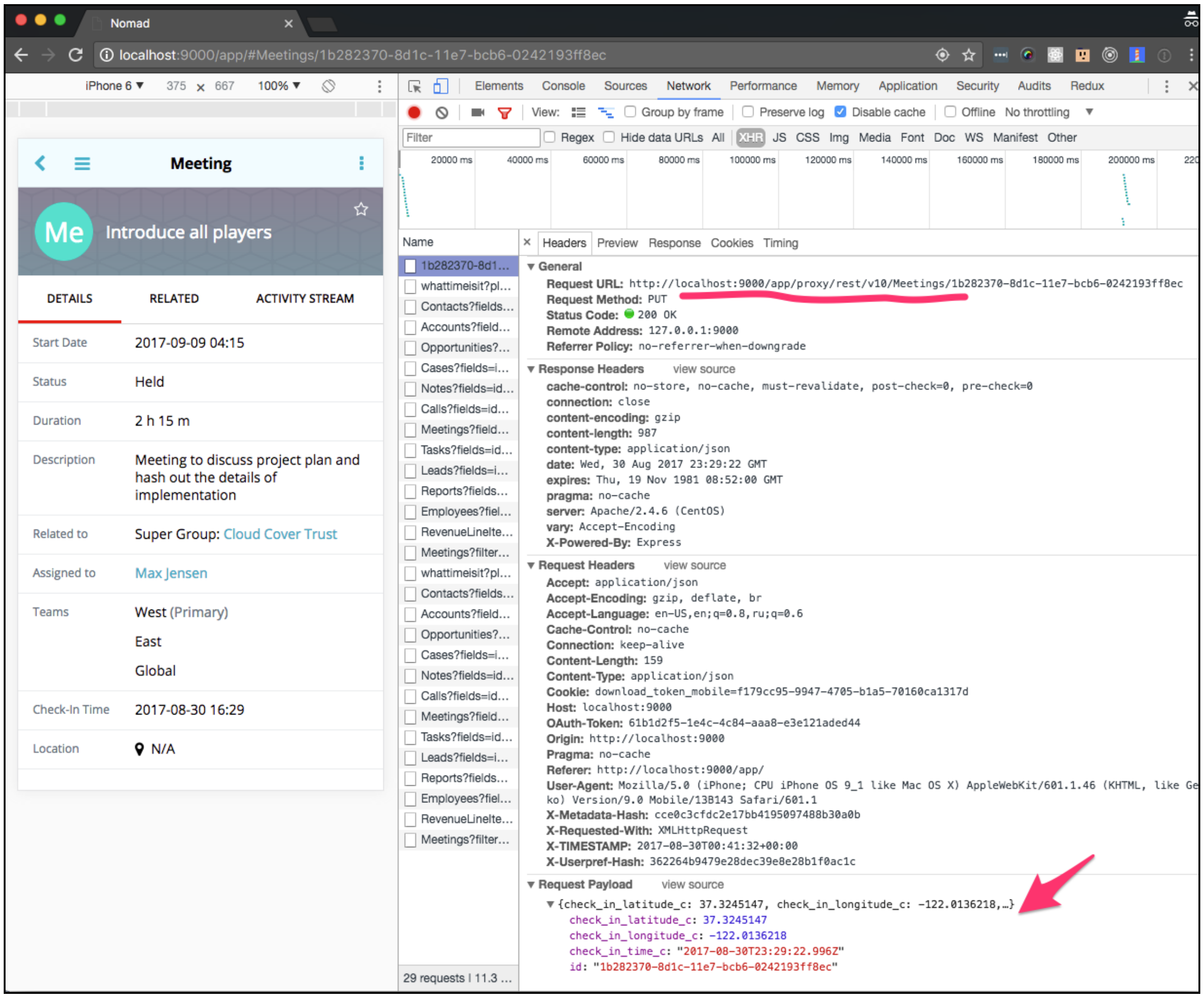
The folder structure should now look like the following:



```
OPEN FILES
FOLDERS
▼ Hero
    ▶ build
    ▶ config
    ▼ custom
        ▼ check-in
              check-in-action.js
        ▼ fields
            ▼ location
                  detail.hbs
                  location.js
        ▶ hello-world
    ▶ native
      .gitignore
      cordova
      get-sdk-path.js
      package.json
      sdk
```

Done! Run the app in a desktop browser to see the action in action (pun intended). Navigate to a meeting record detail view (you may need to create a Meetings record if your Sugar instance doesn't have any meetings) and click the **Check-In** button on the toolbar. Your browser should ask your permission to retrieve your location; click **Allow**. Your location will be stored in the **Location** field. Click on it to see your location on Google Maps.

In your browser, check out the developer tools network tab to see the REST API calls. After the check-in action completes successfully, you should see a PUT request that sends your coordinates as well as the current time and date.

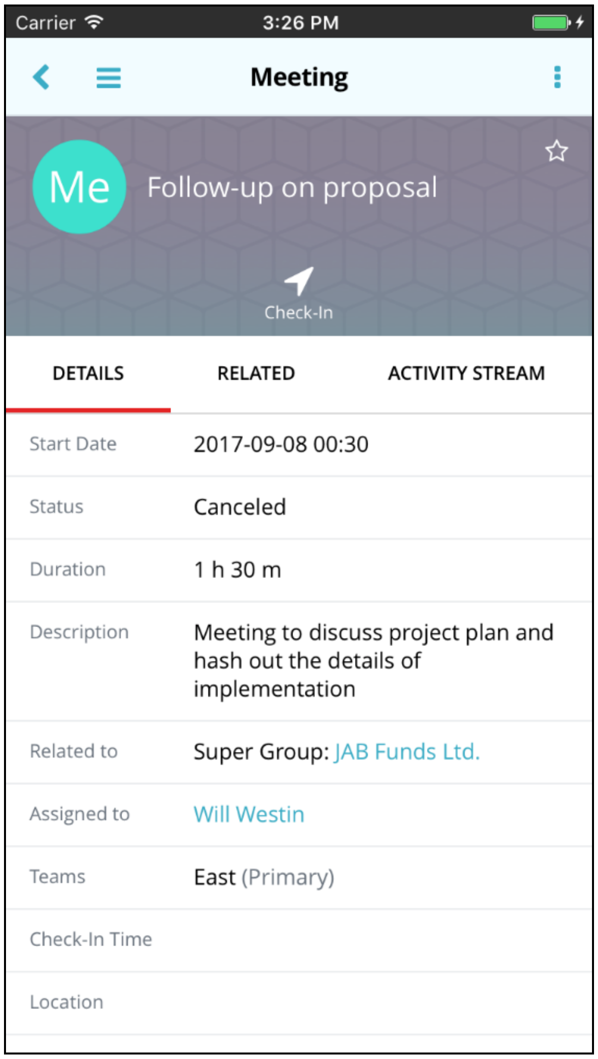

Nomad
localhost:9000/app/#Meetings/1b282370-8d1c-11e7-bcb6-0242193ff8ec
iPhone 6 375 x 667 100%
Elements Console Sources Network Performance Memory Application Security Audits Redux
Meeting
Me Introduce all players
DETAILS RELATED ACTIVITY STREAM
Headers Preview Response Cookies Timing
29 requests | 11.3 ...

Start Date	2017-09-09 04:15
Status	Held
Duration	2 h 15 m
Description	Meeting to discuss project plan and hash out the details of implementation
Related to	Super Group: Cloud Cover Trust
Assigned to	Max Jensen
Teams	West (Primary) East Global
Check-In Time	2017-08-30 16:29
Location	N/A

```
▼ General
  Request URL: http://localhost:9000/app/proxy/rest/v10/Meetings/1b282370-8d1c-11e7-bcb6-0242193ff8ec
  Request Method: PUT
  Status Code: 200 OK
  Remote Address: 127.0.0.1:9000
  Referrer Policy: no-referrer-when-downgrade
▼ Response Headers    view source
  cache-control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
  connection: close
  content-encoding: gzip
  content-length: 987
  content-type: application/json
  date: Wed, 30 Aug 2017 23:29:22 GMT
  expires: Thu, 19 Nov 1981 08:52:00 GMT
  pragma: no-cache
  server: Apache/2.4.6 (CentOS)
  vary: Accept-Encoding
  X-Powered-By: Express
▼ Request Headers    view source
  Accept: application/json
  Accept-Encoding: gzip, deflate, br
  Accept-Language: en-US,en;q=0.8,ru;q=0.6
  Cache-Control: no-cache
  Connection: keep-alive
  Content-Length: 159
  Content-Type: application/json
  Cookie: download_token_mobile=f179cc95-9947-4705-b1a5-70160ca1317d
  Host: localhost:9000
  OAuth-Token: 61b1d2f5-1e4c-4c84-aaa8-e3e121aded44
  Origin: http://localhost:9000
  Pragma: no-cache
  Referer: http://localhost:9000/app/
  User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 9_1 like Mac OS X) AppleWebKit/601.1.46 (KHTML, like Ge
  ko) Version/9.0 Mobile/13B143 Safari/601.1
  X-Metadata-Hash: cce0c3cfdc2e17bb4195097488b30a0b
  X-Requested-With: XMLHttpRequest
  X-TIMESTAMP: 2017-08-30T00:41:32+00:00
  X-Userpref-Hash: 362264b9479e28dec39e8e28b1f0ac1c
▼ Request Payload    view source
  ▼ {check_in_latitude_c: 37.3245147, check_in_longitude_c: -122.0136218,…}
      check_in_latitude_c: 37.3245147
      check_in_longitude_c: -122.0136218
      check_in_time_c: "2017-08-30T23:29:22.996Z"
      id: "1b282370-8d1c-11e7-bcb6-0242193ff8ec"
```

Oh-oh! The address field is empty! That's because desktop browsers don't have reverse geocoding capabilities. Test the native application to see geocoding in action:

1. Update the web bundle for native apps by running `./sdk bundle-web -p native`.
2. Use Xcode or Android Studio to run the native app. Try playing with the check-in action in an emulator or a device.



Follow-up on proposal

Start Date	2017-09-08 00:30
Status	Canceled
Duration	1 h 30 m
Description	Meeting to discuss project plan and hash out the details of implementation
Related to	Super Group: JAB Funds Ltd.
Assigned to	Will Westin
Teams	East (Primary)
Check-In Time	
Location	

Great! You just implemented a custom field and integrated a custom record action that uses the device native API. How about showing check-in locations on a map? Not a problem. Proceed to the bonus assignment.

Bonus Assignment 4: Meetings Map

IMPORTANT: You need to complete Assignment 3 in order to proceed with this assignment.

Sometimes it is impossible to implement certain functionality using web technologies. In such cases, you can integrate a Cordova plugin. There are many 3rd-party open source plugins. You can also implement your own plugin from scratch.

In this exercise we will integrate a 3rd-party Google Maps plugin.

1. First of all, the Google Maps API requires API keys. You can obtain them from the Google Maps developer portal at https://developers.google.com/maps/.

2. Second, install the plugin into the native app project. Open the terminal window and run the following commands:

➔ `cd native`

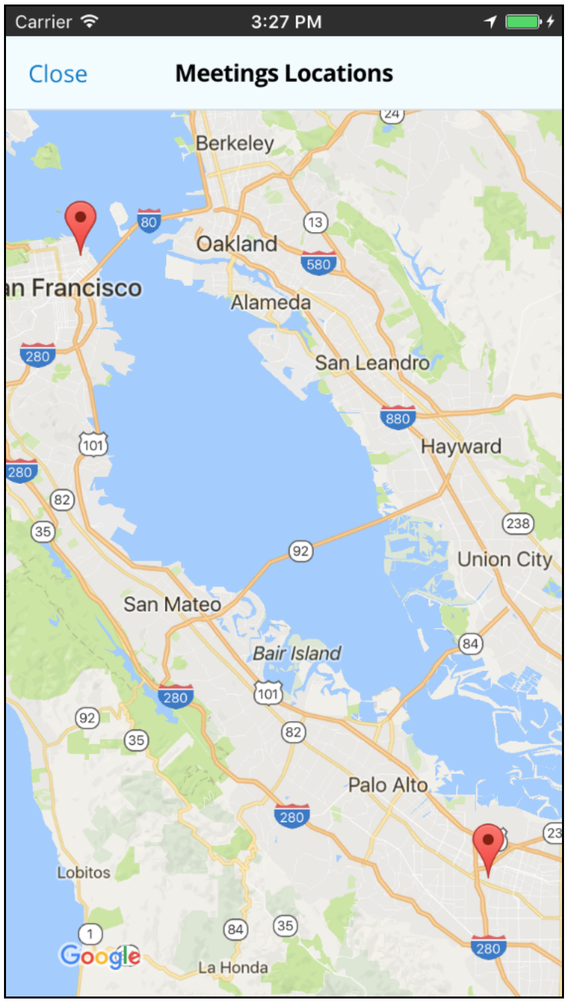
➔ `.../cordova plugin add https://github.com/nomadians/cordova-plugin-googlemaps#v1_4_5 --variable API_KEY_FOR_ANDROID="your android api key" --variable API_KEY_FOR_IOS="your ios api key"`

4. Implement the Meetings map view.
Copy `meetings-map-view.js` and `meetings-map.hbs` from http://bit.ly/mobilesdk_assign4 to your local `Hero/custom/check-in` folder. Check out the JavaScript controller. It contains the declaration of the `MeetingsMapView` class. This is the view that will render the Google map with markers of meeting locations.

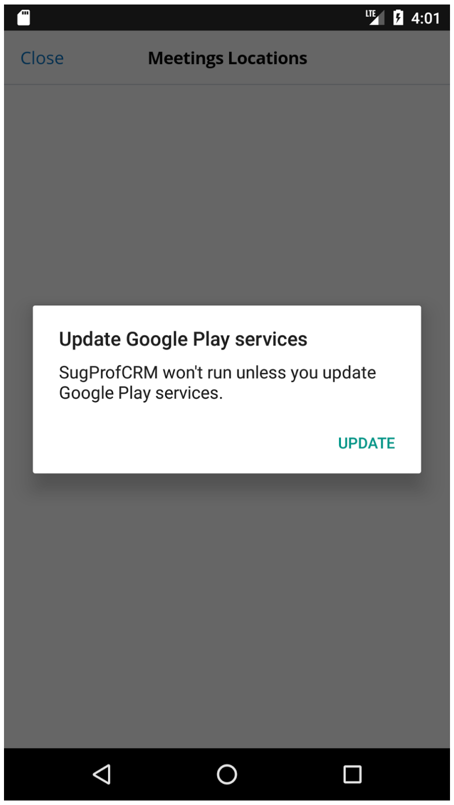
The controller also contains the definition of a custom "map" action. We register the action as an item for the right menu on the Meetings list view. Choosing this menu item loads `MeetingsMapView` as a so-called dynamic view, i.e. we didn't have to define a navigation route for it.

5. Rebuild the web bundle and run the app from Xcode or Android Studio.

```
./sdk bundle-web -p native
```



NOTE: If you want to run the app on Android Emulator, use Nexus 5X or Nexus 5 based on “Google Play Intel x86 Atom System Image”. This image is available for Android 7.0 and 8.0. You may be prompted to update “Google Play services” on the first attempt to show the map view. Click Update, then close and reopen the map view:



Update Google Play services

SugProfCRM won't run unless you update Google Play services.

Extra credit

Wanna keep going? Here are some ideas:

- Run the app on a tablet device to see what the custom map view looks like.
- Render the map view on the left and meeting DetailView on the right or
- Try to render the map view on the right next to the meetings ListView.

Summary

Congratulations! Now you know how to customize the SugarCRM Mobile app. You just learned how to:

- Brand and style the app.
- Implement a simple custom view.
- Add a custom record action.
- Use geolocation services.
- Integrate native functionality.

You can find the complete solution in the Uncon github repository at https://github.com/sugarcrm/uncon/tree/2017/mobile_sdk/Hero.

Share your success

We'd love to see your progress! Tweet a screenshot of your app or a selfie of yourself with your favorite Sugar tutorial expert. Here are some ideas to get you started:

- Built a mobile app in less than a day! #Uncon #SugarCon
- Sugar Mobile SDK rocks! #Uncon #SugarCon

If you continue on with the extra credit, take this tutorial in a new direction, or have feedback, we'd love to hear about it! Email us at developers@sugarcrm.com.

Additional resources

- Sugar Mobile Developer Portal Documentation: mobiletools.sugarcrm.com/#/download-sdk
- Full source code of the Hero app: github.com/sugarcrm/uncon/tree/2017/mobile_sdk/Hero

Need help?

Post your questions in the UnCon Community: https://community.sugarcrm.com/community/uncon.